S

Scala

Alexander Alexandrov
Database Systems and Information Management Group (DIMA), Technische Universität Berlin, Berlin, Germany

Definitions

Scala is a statically typed General-purpose Programming Language (GPL) which blends object-oriented and functional programming features. Scala source code compiles to bytecode that runs on the Java Virtual Machine (JVM) and is fully interoperable with code written in Java.

Overview

Scala was designed by Martin Odersky at the École polytechnique fédérale de Lausanne (EPFL) in 2001 and first released in 2003 (Odersky 2006). The original design goal was to combine features from object-oriented and functional programming in a programming language that can be used in practice. The following paragraphs summarize the main features of Scala in comparison to Java.

Objects and Classes

Scala distinguishes between classes – which can be instantiated multiple times using the `new` keyword (as in Java) – and objects, which can be understood as a singleton instance of an implicitly derived class definition. An object and a class that share the same name are called companions. Methods and fields that would be declared static in Java must be defined within the companion object of the corresponding class in Scala. For example, the `bar` method in the following code snippet corresponds to a static method declaration in Java.

```
class Foo {
  ...
}
object Foo {
  def bar(): Unit = ...
}
```

Scala also supports Java-like abstract classes using the `abstract` keyword.

Inheritance and Traits

Instead of interfaces, Scala relies on the concept of traits. Similar to a Java interface, a Scala trait defines methods and fields that can be attached to an object or a class by means of inheritance. In contrast to Java interfaces, however, traits (*a*) can be partially implemented (which is also possible in Java since version 8) and (*b*) allow for multiple inheritance – an object or a class can inherit from multiple traits at the same time. The following code snippet defines a class `Cat` which inherits from `Animal`, `Eyes`, and `Mouth`. `Cat` instances therefore can call the `blink` and `eat` methods defined in their parent traits.

S. Sakr, A. Y. Zomaya (eds.), *Encyclopedia of Big Data Technologies*,
https://doi.org/10.1007/978-3-319-77525-8

```
abstract class Animal
trait Eyes {
  def blink(): Unit = ...
}
trait Mouth {
  def eat(f: Food): Unit = ...
}
class Cat extends Animal
  with Eyes
  with Mouth
```

Type System

Scala's type system offers more principled support for generics with declaration-site variance, whereas Java allows only for use-site variance. For example, since `Cat` inherits from `Animal`, a covariant type parameter `A` in the generic type `List` (indicated with `+`) allows for assigning instances of type `List[Cat]` to a variable of type `List[Animal]`.

```
class List[+A] { ... }
var xs: List[Animal] = null
xs = List.empty[Cat]
```

In addition, Scala also supports type members and path-dependent types. For example, the following trait defines a trait for comparators for an abstract element type `T`

```
trait Cmp {
  type T
  def cmp(a: T, b: T): Int
}
```

and the `sort` method below accepts a comparator `c` and a sequence of comparable elements of type `c.T` and returns them as a sorted sequence.

```
def sort
  (c: Cmp)
  (s: Seq[c.T]): Seq[c.T]
```

The type `c.T` in the method declaration is *path-dependent* as it depends on the parameter value `c`.

Implicits

Scala implicits encompasses a range of language features that enable a number of idiomatic encodings.

Implicit method parameters can be omitted from method applications. The Scala compiler automatically provides arguments for implicit parameters from the set of implicit values available at the call site. For example, the `sort` method from the previous section can be also declared as follows.

```
def sort[T]
  (s: Seq[T])
  (implicit c: Ord[T]): Seq[T]
```

The above declaration permits incomplete method calls such as

```
sort(Seq(3,4,1,5))
```

if an implicit value of type `Ord[T]` is available in the surrounding lexical scope. A shorthand notation which expands to the above method definition can be used to encode F-bounded polymorphism (Canning et al. 1989) in Scala.

```
def sort[T: Ord]
  (s: Seq[T]): Seq[T]
```

While an unconstrained type parameter `T` corresponds to universal quantification, the `Ord` type constraint corresponds to existential quantification: for all types `T`, the existence of an instance of type `Ord[T]` implies that given a value of type `Seq[T]` we can produce a sorted value of type `Seq[T]`.

TODO

Implicit classes provide a language facility for automatic type conversion. An idiomatic use of this feature is the so-called "pimp my library" pattern. It allows to attach methods and fields to types provided by external libraries in an ad hoc manner. To illustrate the pattern, assume that the `Cat` type from the previous section is provided by an external library that we cannot modify, but we want to add walking facility to `Cat` instances. Implicit classes allow us to achieve that as follows.

```
implicit class CatOps(c: Cat) {
  def walk(): Unit = ...
}
val tom = new Cat
tom.walk()
```

Because the `CatOps` class is declared as `implicit`, the Scala compiler will

automatically wrap the `Cat` instance in the `tom.walk()` call into a `CatOps` constructor call.

```
new CatOps(tom).walk()
```

Implicit conversions are another feature that can be used to encode the "pimp my library" pattern. To that end, instead of declaring the `CatOps` class as implicit, we define an `implicit` method called `cat2ops` which converts a `Cat` instance into a `CatOps` instance.

```
implicit def cat2ops(c: Cat) =
  new CatOps(c)
```

The expanded version of `tom.walk()` now uses the `cat2ops` method instead.

```
cat2ops(tom).walk()
```

Metaprogramming

Metaprogramming is the ability of a programming language to reflect on its own terms and types, manipulate those, and generate new terms and types. Since version 2.10, Scala ships with experimental support for compile-time and run-time metaprogramming in the form of Scala macros (Burmako 2013) and Scala reflection (Coppel et al. 2008). These metaprogramming facilities are extensively used to reduce boilerplate code when designing Domain Specific Languages (DSLs) embedded in Scala, especially in combination with F-bounded polymorphism.

Actor Model and Akka

Akka is a widely used third-party Scala library that implements the actor model for concurrent computation. In the actor model, computation is modeled by actors which exchange messages in an asynchronous manner. Akka hides specifics regarding actor placement behind a uniform API. This allows programmers to run Akka code within same process, within multiple processes on same machine and on different machines without any modifications.

Key Research Findings

From a programming language perspective, the essence of Scala's type system has been recently formalized in terms of the Dependent Object Types (DOT) calculus and proven sound using a mechanized proof by Rompf and Amin (2016). An influential line of research in Scala-based Embedded Domain Specific Languages (eDSLs) is based on the Lightweight Modular Staging (LMS) framework by Rompf and Odersky (2010). In the domain of Big Data analytics, LMS has been used in the Delite framework (Sujeeth et al. 2014) and a DSL called Jet (Ackermann et al. 2012).

From a data management perspective, Scala has been successfully used as implementation language for a number of research systems for Big Data analytics, most notably Spark (Zaharia et al. 2010) and Stratosphere/Flink (Alexandrov et al. 2014).

Examples of Application

Due to its concise and flexible syntax and its built-in interactive shell, Scala is a popular choice for a host language for eDSLs. In the domain of Big Data analytics, Scala has been popularized by Apache Spark and its Resilient Distributed Dataset (RDD) and Dataset/DataFrame DSLs (Armbrust et al. 2015; Zaharia et al. 2010), by Apache Mahout and its Samsara DSLs (Schelter et al. 2016), and by the Scala DSLs offered by systems such as Summingbird (Boykin et al. 2014) and Apache Flink (Carbone et al. 2015).

Embedded DSLs can be either shallow or deep (Gibbons and Wu 2014). In shallow embedding, eDSL terms evaluate themselves directly at run-time. Contrary, deeply embedded DSLs evaluate themselves in a two steps, first reflecting on their structure in an Intermediate Representation (IR) and then interpreting (and possibly optimizing) this IR. The DSLs mentioned above can be classified as deep, and their embedding mechanism as type-based. The following paragraphs illustrate how the Scala features outlined above are

commonly applied in these DSLs based on their implementation methodology.

Type-based DSLs are structured around a collection of domain-specific types, realized as Scala classes with possible companion objects. The core types in Spark are `RDD` (in the RDD DSL) and `Dataset` (in the Dataset/DataFrame DSL) which represent a distributed collection managed by the Spark runtime. The core types in Apache Mahout are `Matrix` which represents an in-core matrix and `DrmLike` which represents a Distributed Row Matrix (DRM). The core type in Summingbird is `Producer`, which represents a streaming or batch data producer. Finally, the core types for the various DSLs exposed by Apache Flink are `DataSet`, `DataStream`, or `Table`.

Implicit classes and conversions are often used in the design of type-based DSLs in order to provide ad hoc methods for a specific kind of instances of the associated core DSL types. For example, RDDs in Spark offer a set of operators that are only available on an RDDs of key-value pairs. The implicit method `rddToPairRDDFunctions` converts value of type `RDD[(K,V)]` into a value of an `RDDOps`-like type `PairRDDFunctions[K,V]`. This allows to leverage the existing Scala infrastructure in order to statically ensure that expressions such as

```
val rdd: RDD[Int, Int] = ...
rdd.countByKey()
```

are valid, while expressions such as

```
val rdd: RDD[Int] = ...
rdd.countByKey()
```

fail with a Scala type error due to a missing implicit conversion.

Embedded DSLs for data-intensive analytics hosted in Scala also rely on Scala's metaprogramming facilities, usually in conjunction with F-bounded polymorphism. For example, in order to reduce the pressure on the JVM garbage collector, the `Dataset` DSL in Spark and the DSLs offered by Flink rely on engine-specific memory management on serialized data. This means that for a distributed collection of type `Dataset[A]` (in Spark) or `DataSet[A]` (in Flink), the runtime needs to know how to serialize and deserialize instances of the generic element type `A`. To ensure that, every DSL method that changes the element type of the enclosing distributed collection is constrained by an associated type that provides the encoding and decoding functionality for the new element type. For example, the `map` method in Spark is constrained by an `Encoder[B]` instance

```
def map[B: Encoder](
  f: A => B
): Dataset[B]
```

and the corresponding method in Flink by a `TypeInformation[B]` instance.

```
def map[B: TypeInformation](
  f: A => B
): Dataset[B]
```

The frameworks also provide generic implementations that can implicitly synthesize `Encoder[A]` and `TypeInformation[A]` instances based on reflected type information about the type argument `A`. Spark's implementation is based on Scala's runtime reflection library, while Flink's implementation is based on Scala macros.

Future Directions for Research

In the area of core language development, current and future research in Scala is mostly related to the next-generation Scala compiler called Dotty developed at the EPFL (Odersky et al. 2016, 2018; Odersky 2006). The Dotty design aims for a combination of faster compilation times and more principled and sound language design based on the DOT calculus.

In the area of DSL development, future research aims to provide a more stable foundation and better tooling for rapid development of optimizing DSLs embedded in Scala. An important milestone in this direction is the Squid metaprogramming framework by Parreaux et al.

(2018). Squid combines the flexibility of dynamic quasiquotes (in the style offered by Lisp) with the typing and scoping guarantees of static quasiquotes (in the style offered by MetaML).

Cross-References

- Apache Flink
- Apache Kafka
- Apache Mahout
- Apache Spark
- Spark SQL

References

Ackermann S, Jovanovic V, Rompf T, Odersky M (2012) Jet: an embedded DSL for high performance big data processing. In: International workshop on end-to-end management of Big Data (BigData 2012), EPFL-CONF-181673

Alexandrov A, Bergmann R, Ewen S, Freytag J, Hueske F, Heise A, Kao O, Leich M, Leser U, Markl V, Naumann F, Peters M, Rheinländer A, Sax MJ, Schelter S, Höger M, Tzoumas K, Warneke D (2014) The stratosphere platform for big data analytics. VLDB J 23(6):939–964. https://doi.org/10.1007/s00778-014-0357-y

Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, Zaharia M (2015) Spark SQL: relational data processing in spark. In: Sellis TK, Davidson SB, Ives ZG (eds) Proceedings of the 2015 ACM SIGMOD international conference on management of data, 31 May–4 June 2015. ACM, Melbourne, pp 1383–1394. https://doi.org/10.1145/2723372.2742797

Boykin PO, Ritchie S, O'Connell I, Lin JJ (2014) Summingbird: a framework for integrating batch and online mapreduce computations. PVLDB 7(13):1441–1451

Burmako E (2013) Scala macros: let our powers combine! On how rich syntax and static types work with metaprogramming. In: Proceedings of the 4th workshop on scala, SCALA@ECOOP, 2 July 2013. ACM, Montpellier, pp 3:1–3:10. https://doi.org/10.1145/2489837.2489840

Canning PS, Cook WR, Hill WL, Olthoff WG, Mitchell JC (1989) F-bounded polymorphism for object-oriented programming. In: Stoy JE (ed) Proceedings of the fourth international conference on functional programming languages and computer architecture, FPCA, 11–13 Sept 1989. ACM, London, pp 273–280. https://doi.org/10.1145/99370.99392

Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache flink™: stream and batch processing in a single engine. IEEE Data Eng Bull 38(4):28–38

Coppel Y, Odersky M, Dubochet G (2008) Reflecting scala. Semester project report, Laboratory for Programming Methods Ecole Polytechnique Federale de Lausanne, Lausanne

Gibbons J, Wu N (2014) Folding domain-specific languages: deep and shallow embeddings (functional pearl). In: ICFP. ACM, pp 339–347

Odersky M (2006) A brief history of scala. Blog Post. http://www.artima.com/scalazine/articles/origins_of_scala.html

Odersky M, Martres G, Petrashko D (2016) Implementing higher-kinded types in dotty. In: Biboudis A, Jonnalagedda M, Stucki S, Ureche V (eds) Proceedings of the 7th ACM SIGPLAN symposium on scala, SCALA@SPLASH 2016, 30 Oct–4 Nov 2016. ACM, Amsterdam, pp 51–60. https://doi.org/10.1145/2998392.2998400

Odersky M, Blanvillain O, Liu F, Biboudis A, Miller H, Stucki S (2018) Simplicitly: foundations and applications of implicit function types. PACMPL 2(POPL):42:1–42:29. https://doi.org/10.1145/3158130

Parreaux L, Voizard A, Shaikhha A, Koch CE (2018) Unifying analytic and statically-typed quasiquotes. PACMPL 2(POPL):13:1–13:33. https://doi.org/10.1145/3158101

Rompf T, Amin N (2016) Type soundness for dependent object types (DOT). In: Visser E, Smaragdakis Y (eds) Proceedings of the 2016 ACM SIGPLAN international conference on object-oriented programming, systems, languages, and applications, OOPSLA 2016, part of SPLASH 2016, 30 Oct–4 Nov 2016. ACM, Amsterdam, pp 624–641. https://doi.org/10.1145/2983990.2984008

Rompf T, Odersky M (2010) Lightweight modular staging: a pragmatic approach to runtime code generation and compiled DSLs. In: Visser E, Järvi J (eds) Generative programming and component engineering, proceedings of the ninth international conference on generative programming and component engineering, GPCE 2010, 10–13 Oct 2010. ACM, Eindhoven, pp 127–136. https://doi.org/10.1145/1868294.1868314

Schelter S, Palumbo A, Quinn S, Marthi S, Musselman A (2016) Samsara: declarative machine learning on distributed dataflow systems. In: NIPS workshop ML-Systems

Sujeeth AK, Brown KJ, Lee H, Rompf T, Chafi H, Odersky M, Olukotun K (2014) Delite: a compiler architecture for performance-oriented embedded domain-specific languages. ACM Trans Embedded Comput Syst 13(4s):134:1–134:25. https://doi.org/10.1145/2584665

Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. In: Nahum EM, Xu D (eds) 2nd USENIX workshop on hot topics in cloud computing, HotCloud'10, 22 June 2010. USENIX Association, Boston

Scalable Architectures for Big Data Analysis

Peng Sun and Yonggang Wen
School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore

Overview

The era of big data is upon us. However, traditional data management and analysis systems, which are mainly based on relational database management system (RDBMS), may not be able to handle the ever-growing data volume. Therefore, it is important to design scalable system architectures to efficiently process big data and exploit their value. This chapter discusses various horizontal and vertical scaling big data platforms, focusing on their architectural principle for big data analysis applications, such as machine learning and graph processing. This chapter could aid users to select right system architectures or platforms for their big data applications.

Introduction

This is an era of big data, evidenced by the sheer volume of data from a variety of sources and its growing rate of generation. According to a report from the International Data Corporation (IDC), the global data volume will grow by a factor of 300, from 130 exabytes (1 exabyte = 10^6 terabytes) to 40,000 exabytes, from 2005 to 2020 (Gantz and Reinsel 2007). These data come from everywhere (Hu et al. 2014), such as user-generated contents (e.g., images and videos) posted to social media sites, transaction records, and embedded sensors used to gather information for IoT (Internet of Things) applications.

The concept of big data has been defined as early as 2001. META group (now Gartner) analyst Doug Laney presented a "3Vs" model (Laney 2001) to define challenges and opportunities of data growth, i.e., increasing volume, velocity, and variety. Although this description was not originally used to define big data, many industries continue to use this "3Vs" model to describe big data 10 years later (Chen et al. 2014). In 2012, Gartner updated the definition of big data as follows: "*Big data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization*" (Beyer and Laney 2012).

Traditional data management and analysis systems are mainly based on the relational database management system (RDBMS), which could not efficiently handle big data. Specifically, RDBMSs could only support structured data, while big data typically includes masses of semi-structured and unstructured data. Also, the ever-growing data volume could easily extend beyond the ability of average RDBMSs to capture, store, manage, and analyze big data.

To address these challenges and exploit the value of big data, the research community and industry have proposed various scalable big data analysis platforms. Scalability is the ability of a data analysis system to process increasing amounts of data in an appropriate manner. Typically, a big data analytic system should be able to support very large datasets and could be capable of scaling to address the ever-growing size of complex datasets generated in the future (Hu et al. 2014).

This article discusses various big data analysis platforms and focuses on the scalability and the architectural design. Specifically, we categorize existing big data analysis platforms into two types of scaling: horizontal and vertical scaling. For each type of scaling, we illustrate the architectural principle of several representative general-purpose and dedicated big data platforms. Moreover, this article also discusses the advantages and drawbacks of horizontal and vertical scaling to aid users to select the appropriate platforms for specific big data applications or algorithms.

The rest of this article is organized as follows. Section "Scalability in Big Data Analysis" presents the fundamental concept of scalability for big data analytics. Section Horizontal Scaling

Platforms introduces various horizontal scaling big data analysis platforms, including general-purpose systems like Hadoop and Spark and dedicated systems for machine learning and graph processing. Section "Vertical Scaling Platforms" describes vertical scaling big data analysis platforms using graphics processing units (GPUs) and field-programmable gate arrays (FPGAs). A brief conclusion with recommendations for future studies is presented in section "Summary".

Scalability in Big Data Analysis

To capture, manage, and analyze exploding datasets, scaling has become a key technique for big data analysis. A scalable big data analysis platform could adapt to rapid changes in the growth of data. Different platforms incorporate different scaling techniques to support big data analysis.

Horizontal and Vertical Scaling

Popular big data platforms usually use the following two types of scaling to handle big data:

- **Horizontal Scaling:** Horizontal scaling, which is also known as "scaling out," involves distributing the computation workload across a cluster with multiple commodity servers. With this approach, a big data analysis platform could easily improve its processing capability by adding independent machines. Typically, each machine runs its own operating system and could communicate with other machines via network to exchange intermediate data and controlling information for big data analysis.
- **Vertical Scaling:** Vertical scaling, which is also known as "scaling up," involves adding more resources, such as processors and memory, to a single server for improving its data processing capability. With this approach, a big data analysis job runs in a single machine with a single operating system. Note that vertical scaling is often done through multi-threading computation and in-process message passing.

Comparison of Horizontal and Vertical Scaling

Table 1 compares the advantages and drawbacks of horizontal and vertical scaling. Most of the existing single-node data analysis tools could take advantage of faster hardware to improve the processing capability. However, it is hard to continuously scale up vertically after a certain limit (Hu et al. 2014). As a comparison, horizontal scaling could continuously increase system performance by installing new machines. Note that network could easily be the performance bottleneck for big data analysis when scaling out. Horizontal scaling big data analysis platforms must take care of this issue to reduce the communication overhead.

The initial investment costs of horizontal scaling big data platforms are relatively less than vertical scaling platforms. Specially, users could build a small commodity cluster to deal with the current big data analysis workload and install more machines to handle exploding datasets generated in the future. As a comparison, when a data analysis system is designed for vertical scaling only, users are locked into certain minimum initial investment costs driven by the hardware, which could handle current and future workload.

Regarding maintenance costs, vertical scaling big data platforms have an advantage than horizontal scaling platforms. Specifically, when users select vertical scaling platforms, they only need to cool a single server. As a compression, when using horizontal scaling platforms with multiple servers and network components, the cooling system may consume a huge amount of energy, which could significantly increase the maintenance costs (Dayarathna et al. 2016).

Horizontal Scaling Platforms

Many horizontal scaling platforms have been proposed for big data analysis. Message Passing Interface (MPI), MapReduce, and Spark are thee widely used general-purpose big data platforms. There are also a lot of dedicated platforms that are proposed for machine learning or graph processing.

Scalable Architectures for Big Data Analysis, Table 1 Advantages and drawbacks of horizontal and vertical scaling for big data analysis

Scaling	Advantages	Drawbacks
Horizontal scaling	• Increase processing capability by adding machines	• Network could easily be the performance bottleneck
	• Could avoid the single point of failure problem	• May have high power consumption to cool a cluster
	• Could handle big data with commodity machines	• Time-consuming to configure multiple machines
Vertical scaling	• Existing data analysis tools and softwares could easily take advantage of faster hardware in a single machine	• High financial investment to improve the processing capability of a single-node data analysis system
	• Easy to manage the hardware within a single machine	• Hard to scale up vertically after a certain limit
	• Lower power consumption with just one machine	• May have the single point of failure problem

General-Purpose Big Data Analysis Platforms

MPI

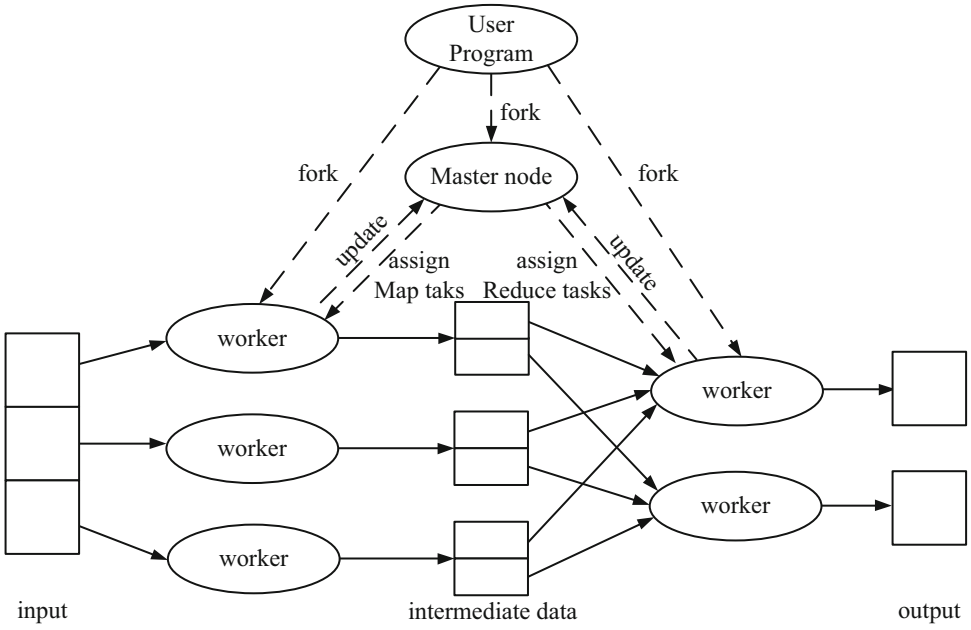
MPI (Gropp et al. 1999) is a standard communication protocol for programming parallel computers using the peer-to-peer architecture. Typically, MPI is used to set up the communication channel between different computation nodes. Thus, each node could exchange data and control information with each other to execute big data analysis applications. MPI is available for many programming languages and has several well-tested and efficient implementations, such as Open MPI (Gabriel et al. 2004), MPICH (Karonis et al. 2003) and MVAPICH (Panda et al. 2013).

MPI supports both point-to-point and collective communication. More specifically, point-to-point communication is the method to send and receive messages between two individual nodes. Collective communication is a method of communication which involves the participation of all selected nodes. For example, "Broadcast," one of the standard collective communication techniques, sends the same data to all selected nodes. MPI also provides a set of methods to control the progress of each node. For example, the "Barrier" method puts a barrier and allows all computation nodes to synchronize (reaching up to a certain point) before proceeding further. With these APIs, users could implement their own big data analysis applications and run them in an MPI-enabled cluster with the specified amount of resources.

Although MPI offers high flexibility for users to develop big data analysis applications, it has some major drawbacks. First, MPI has no mechanism to handle faults. Thus, when running MPI-based big data analysis applications with unreliable hardware (e.g., commodity machines), a single-node failure could cause the whole application to crash. Thus, it is users' responsibility to design and implement fault-tolerance mechanism when processing big data with MPI. Second, MPI requires more programming changes to design and implement a distributed data analysis application. Hadoop MapReduce and Spark have been designed to addresses these challenges and become popular for big data analysis.

Hadoop MapReduce

Apache Hadoop (White 2012) is an open source project for storing, managing, and processing large-scale datasets using clusters of commodity machines. Hadoop is designed to scale out to hundreds or even thousands of computation nodes. Hadoop contains various components, including Hadoop Distributed File System (HDFS) for reliable big data storage, Hadoop Yarn for efficient cluster resource management, and Hadoop MapReduce for big data analysis.



Scalable Architectures for Big Data Analysis, Fig. 1 Mapreduce architecture

MapReduce, which was firstly proposed by Google (Dean and Ghemawat 2008), is the programming model used in Hadoop. As shown in Fig. 1, there are two types of nodes in a MapReduce cluster: master node and worker. Specifically, the master node (i.e., JobTracker) is in charge of scheduling the submitted MapReduce jobs: partitioning each job into two types of tasks (i.e., Map tasks and Reduce tasks) and assigning them to available workers for execution. Current Hadoop MapReduce uses slots, which is usually configured by the number of CPU cores, to partition the computation resources of a worker. For example, if one server is configured to have two Map slots and two Reduce slots, it could run two Map tasks and two Reduce tasks in parallel.

To perform big data analysis, users just need to write two functions: Map function and Reduce function. When a job is submitted, the underlying MapReduce execution engine will automatically parallelize, distribute, and execute the job on a cluster and process data with two phases, Map phase and Reduce phase, as shown in Fig. 1. Specifically, during the Map phase, workers read input, generate a number of intermediate data, and send them to corresponding workers. During the Reduce phase, workers process each group of intermediate data in parallel and generate the final results.

One of the major drawbacks of MapReduce is its inefficiency for executing iterative algorithms. Specifically, when executing iterative algorithms with MapReduce, workers would read the same input data from disks at the beginning of each iteration. After each iteration, the results should be written into disks, which may be loaded into memory again in the following iteration. As a result, disk access could be a major performance bottleneck, which significantly degrades the data processing performance. To address this problem, HaLoop (Bu et al. 2010) and Twister (Ekanayake et al. 2010) extend Hadoop MapReduce and improve the performance of executing iterative algorithms by caching input, output, and intermediate data.

Spark

Spark (Zaharia et al. 2012) was developed in 2012 in response to limitations in MapReduce. It offers a programming model similar to MapReduce but extends it with a data-sharing abstraction called resilient distributed datasets (RDDs).

RDDs are fault-tolerant, parallel data structures across multiple machines, which let users persist input, output, and intermediate results in memory (if the data does not fit in memory, RDD would also spill it to disk), control their partitioning to optimize data placement, and manipulate them using a rich set of operators. With the help of RDD, Spark could eliminate the disk I/O overhead of Hadoop MapReduce for running iterative big data analysis applications.

The Spark developers have also proposed the Berkeley Data Analytics Stack (BDAS) as an open source software stack to make sense of big data. In BDAS, Alluxio (formerly Tachyon) is a reliable, memory speed, distributed storage system. It could help to reduce data access overhead for big data analysis platforms. BDAS contains a cluster resource management systems called Mesos (Hindman et al. 2011), which could efficiently share cluster resources with multiple distributed computing systems. Spark is the processing engine of BDAS. BDAS also contains many Spark wrappers to optimize the performance Spark for certain applications. For example, Spark Streaming (Zaharia et al. 2013) is used for large-scale stream processing. GraphX (Gonzalez et al. 2014) is proposed for high-performance distributed graph processing. MLlib (Meng et al. 2016) focuses on distributed machine learning.

Dedicated Big Data Analysis Platforms

Many big data analysis platforms have been proposed for dedicated purposes, such as machine learning and graph processing. Compared to aforementioned general-purpose platforms like Hadoop MapReduce and Spark, these dedicated platforms usually could offer much higher performance for certain applications or algorithms.

Machine Learning

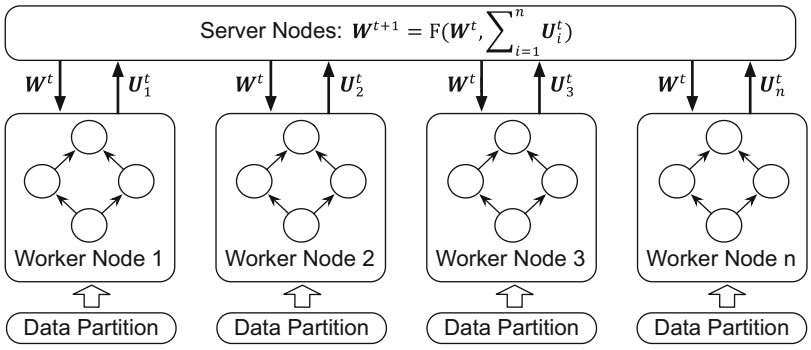
Machine learning (ML) builds models from training data and use them to make predictions on new data. It is used in a wide range of applications, including image recognition, object detection, natural language processing, and recommender systems. Typically, an ML model consists of a large number of *parameters*, represented as dense or sparse vectors and matrices. To minimize the prediction error, an ML application usually uses an iterative-convergent algorithm to iteratively compute *updates* from training datasets and to aggregate *updates* with the current version of *parameters*.

In industrial ML applications, the size of training data sets could be hundreds to thousands of terabytes. For example, the Yahoo News Feed dataset stands at roughly 110 billion lines of user-news interaction data; ImageNet contains approximately 14 million labeled images; ClueWeb12 has 733 million web pages. Due to the limitation in storage and computation resource, modern single-node ML systems, such as Caffe (Jia et al. 2014) and Theano (Bergstra et al. 2011), cannot cope with such large training datasets.

To handle big training datasets, various distributed ML systems have been proposed based on the parameter server (PS) architecture (Li et al. 2014), such as Petuum (Xing et al. 2015), MXNet (Chen et al. 2015b), Project Adam (Chilimbi et al. 2014), SINGA (Wang et al. 2015), Factorbird (Schelter et al. 2014), and TensorFlow (Abadi et al. 2016). As shown in Fig. 2, the PS architecture can scale to large cluster deployments by having *worker* nodes performing data-parallel computation and having *server* nodes maintaining globally shared *parameters*. When training ML models, *worker* nodes continuously pull latest *parameters* from *server* nodes, perform computation on partitions of the training dataset, and push generated *updates* to *server* nodes, which aggregate them and return a new version of *parameters*. This is done iteratively to bring *parameters* closer to the optimal value.

Graph Processing

Graphs are immensely useful for data mining applications, such as social influence analysis, recommendations, clustering, and anomaly detection. As graph sizes increase exponentially, industrial graph processing applications need to process massive graphs at the scale of millions of vertices and hundreds of billions (or even a trillion) edges (Ching et al. 2015). These massive graphs might not fit in the memory of a single



Scalable Architectures for Big Data Analysis, Fig. 2 The PS architecture. In PS, *worker* nodes are in charge of computing *updates*, and *server* nodes manage globally shared *parameters*. PS uses a pull/push communication model to exchange data between *worker* nodes and *server* nodes

machine and cannot be processed by existing single-node in-memory graph computation systems, such as Ligra (Shun and Blelloch 2013) and Ligra+ (Shun et al. 2015).

To address single node's memory limit problem, many distributed in-memory graph computation systems have been proposed for fast graph analytics at scale. They usually follow the "think-like-a-vertex" philosophy and abstract graph computation as vertex-centric programs. Specifically, Pregel (Malewicz et al. 2010), Giraph (Ching et al. 2015), Pregel+ (Yan et al. 2014), GPS (Salihoglu and Widom 2013), MOCGraph (Zhou et al. 2014), and HuSky (Yang et al. 2016) adopt the Pregel model: they assign the input graph's vertices to multiple machines and provide interaction between vertices by message passing along out-edges. PowerGraph (Gonzalez et al. 2012), PowerLyra (Chen et al. 2015a), GraphX (Gonzalez et al. 2014), and LiGraph (Zhao et al. 2016) use the Gather-Apply-Scatter (GAS) model: they split a vertex into multiple replicas and could parallelize the computation for a single vertex in different machines. GraphPad (Anderson et al. 2016) and CombBLAS (Buluç et al. 2011) express common graph analyses in generalized sparse matrix-vector multiplication (SpMV) operations and leverage high-performance computing (HPC) techniques to speed up large-scale SpMV. These distributed in-memory approaches, including Pregel-based, GAS-based, and SpMV-based systems, proceed in iterations, where an iteration is called a superstep. In each superstep, several user-based function (UDFs) are called by active vertices to update their values. A lot of benchmarking results (Hu et al. 2014; Iosup et al. 2016) have shown that these dedicated distributed in-memory systems could offer much better performance than general-purpose cluster computing systems like Hadoop MapReduce and Spark.

To further reduce memory footprint of distributed graph processing, researchers have proposed several out-of-core systems to enable large-scale graph processing beyond the memory limit, such as GraphD (Yan et al. 2017), Pregelix (Bu et al. 2014), and Chaos (Roy et al. 2015) which could handle big graph analytics in a small commodity cluster. Typically, these systems manage all or part of vertices in memory and stream edges from disks to reduce memory footprint.

Vertical Scaling Platforms

This section introduces vertical scaling platforms utilizing heterogeneous devices: GPU and FPGA.

GPU

GPUs started out as graphics accelerators. In recent years, due to their massively parallel architecture, researchers started to use graphics adapters for non-graphics applications. In this context, many programming frameworks start to give rise to GPGPU (general-purpose computing

on graphics processing units). For example, in 2006, NVIDIA released the first version of CUDA (Compute Unified Device Architecture), a parallel computing platform and programming model for general-purpose algorithms on GPUs.

Take NVIDIA's Pascal architecture as an example, GPUs are connected to RAM via PCIe. A GPU contains multiple streaming multiprocessors (SM). Each SM consists of several single-precision arithmetic logic units (ALUs) ("CUDA Cores"). Therefore, compared to multi-core CPU, GPU usually contains a large number of processing cores. For example, TITAN X, featuring the NVIDIA Pascal architecture, contains 3584 CUDA cores. In addition, GPU also has its own high-throughput DDR5 memory, which is much faster than typical DDR3 memory. Thus, GPUs could offer much higher performance than typical CPUs when executing certain big data analysis algorithms.

Nowadays, GPU is playing an important role in big data analysis, especially when executing machine learning and graph processing algorithms. In a single server, GPU could help Caffe (Jia et al. 2014) and Theano (Bergstra et al. 2011) to speed up the training process of deep learning models by a factor of up to 100. Gunrock (Wang et al. 2016), Medusa (Zhong and He 2014), and CuSha (Khorasani et al. 2014) enable fast and simple graph processing on GPUs.

FPGA

FPGAs contain arrays of programmable logic blocks and a hierarchy of reconfigurable interconnects, which allow the blocks to be "wired together." The FPGA configuration is specified using a hardware description language, such as VHDL or Verilog. Due to customized hardware, FPGAs can be highly optimized for certain big data analysis applications. One of the drawbacks of FPGAs is the high development cost, since FPGAs require developers to have a low-level knowledge of the hardware.

FPGAs are used in various big data analysis applications. For example, FPGAs now are widely used to accelerate deep convolutional neural networks (Zhang et al. 2015; Qiu et al. 2016; Ovtcharov et al. 2015). Graphgen (Nurvitadhi et al. 2014) and Fpgp (Dai et al. 2016) start to perform vertex-centric graph computation over FPGAs.

Summary

This article surveys various big data analysis platforms and discusses their architectural design and scaling technique. Existing big data platforms could be categorized into two types of scaling: horizontal and vertical scaling. This article illustrates several representative big data analysis platforms for each scaling type and discusses the advantages and drawbacks of these two scaling techniques. This article could aid users to select right platforms for certain applications.

Cross-References

▸ Scalable Architectures for Big Data Analysis

References

Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray DG, Steiner B, Tucker P, Vasudevan V, Warden P, Wicke M, Yu Y, and Zheng X (2016) Tensorflow: a system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI'16). USENIX Association, Savannah

Anderson MJ, Sundaram N, Satish N, Patwary MMA, Willke TL, and Dubey P (2016) Graphpad: optimized graph primitives for parallel and distributed platforms. In: IPDPS. IEEE, pp 313–322

Bergstra J, Bastien F, Breuleux O, Lamblin P, Pascanu R, Delalleau O, Desjardins G, Warde-Farley D, Goodfellow I, Bergeron A et al (2011) Theano: deep learning on GPUs with python. In: NIPS 2011, BigLearning workshop, Granada

Beyer MA, Laney D (2012) The importance of big data: a definition. Gartner, Stamford, pp 2014–2018

Bu Y, Howe B, Balazinska M, Ernst MD (2010) Haloop: efficient iterative data processing on large clusters. Proc VLDB Endow 3(1–2):285–296

Bu Y, Borkar V, Jia J, Carey MJ, Condie T (2014) Pregelix: big(ger) graph analytics on a dataflow engine. Proc VLDB Endow 8(2):161–172

Buluç A, Gilbert JR (2011) The combinatorial blas: design, implementation, and applications. Int J High Perfor Comput Appl 25(4):496–509

Chen M, Mao S, Liu Y (2014) Big data: a survey. Mob Netw Appl 19(2):171–209

Chen R, Shi J, Chen Y, Chen H (2015a) Powerlyra: differentiated graph computation and partitioning on skewed graphs. In: Proceedings of the tenth European conference on computer systems. ACM, p 1

Chen T, Li M, Li Y, Lin M, Wang N, Wang M, Xiao T, Xu B, Zhang C, Zhang Z (2015b) Mxnet: a flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274

Chilimbi T, Suzue Y, Apacible J, Kalyanaraman K (2014) Project Adam: building an efficient and scalable deep learning training system. In: 11th USENIX symposium on operating systems design and implementation (OSDI'14), pp 571–582

Ching A, Edunov S, Kabiljo M, Logothetis D, Muthukrishnan S (2015) One trillion edges: graph processing at facebook-scale. Proc VLDB Endow 8(12):1804–1815

Dai G, Chi Y, Wang Y, Yang H (2016) FPGP: graph processing framework on FPGA a case study of breadth-first search. In: Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays. ACM, pp 105–110

Dayarathna M, Wen Y, Fan R (2016) Data center energy consumption modeling: a survey. IEEE Commun Surv Tutorials 18(1):732–794

Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

Ekanayake J, Li H, Zhang B, Gunarathne T, Bae SH, Qiu J, Fox G (2010) Twister: a runtime for iterative mapreduce. In: Proceedings of the 19th ACM international symposium on high performance distributed computing. ACM, pp 810–818

Gabriel E, Fagg GE, Bosilca G, Angskun T, Dongarra JJ, Squyres JM, Sahay V, Kambadur P, Barrett B, Lumsdaine A et al (2004) Open MPI: goals, concept, and design of a next generation MPI implementation. In: European parallel virtual machine/message passing interface users group meeting. Springer, pp 97–104

Gantz J, Reinsel D (2012) The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far east. IDC iView IDC Analyze Future 2007(2012):1–16

Gonzalez JE, Low Y, Gu H, Bickson D, Guestrin C (2012) Powergraph: distributed graph-parallel computation on natural graphs. OSDI 12(1):2

Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I (2014) Graphx: graph processing in a distributed dataflow framework. In: 11th USENIX symposium on operating systems design and implementation (OSDI'14). USENIX Association, Broomfield, pp 599–613

Gropp W, Lusk E, Skjellum A (1999) Using MPI: portable parallel programming with the message-passing interface, vol 1. MIT press, Cambridge

Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz RH, Shenker S, Stoica I (2011) Mesos: a platform for fine-grained resource sharing in the data center. NSDI 11:22–22

Hu H, Wen Y, Chua TS, Li X (2014) Toward scalable systems for big data analytics: a technology tutorial. IEEE Access 2:652–687

Iosup A, Hegeman T, Ngai WL, Heldens S, Prat-Pérez A, Manhardto T, Chafio H, Capotă M, Sundaram N, Anderson M et al (2016) Ldbc graphalytics: a benchmark for large-scale graph analysis on parallel and distributed platforms. Proc VLDB Endow 9(13):1317–1328

Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T (2014) Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the ACM international conference on multimedia. ACM, pp 675–678

Karonis NT, Toonen B, Foster I (2003) Mpich-g2: a grid-enabled implementation of the message passing interface. J Parallel Distrib Comput 63(5):551–563

Khorasani F, Vora K, Gupta R, Bhuyan LN (2014) Cusha: vertex-centric graph processing on GPUs. In: Proceedings of the 23rd international symposium on high-performance parallel and distributed computing. ACM, pp 239–252

Laney D (2001) 3D data management: controlling data volume, velocity and variety. META Group Res Note 6(70):1–4

Li M, Andersen DG, Park JW, Smola AJ, Ahmed A, Josifovski V, Long J, Shekita EJ, Su BY (2014) Scaling distributed machine learning with the parameter server. In 11th USENIX symposium on operating systems design and implementation (OSDI'14). USENIX Association, Broomfield, pp 583–598

Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM, pp 135–146

Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S et al (20116) MLlib: machine learning in apache spark. J Mach Learn Res 17(1):1235–1241

Nurvitadhi E, Weisz G, Wang Y, Hurkat S, Nguyen M, Hoe JC, Martínez JF, Guestrin C (2014) Graphgen: an fpga framework for vertex-centric graph computation. In: IEEE 22nd annual international symposium on field-programmable custom computing machines (FCCM). IEEE, pp 25–28

Ovtcharov K, Ruwase O, Kim JY, Fowers J, Strauss K, Chung ES (2015) Accelerating deep convolutional neural networks using specialized hardware. Microsoft Res Whitepaper 2(11):1–4

Panda DK, Tomko K, Schulz K, Majumdar A (2013) The MVAPICH project: evolution and sustainability of an open source production quality MPI library for HPC. In: Workshop on sustainable software for science: practice and experiences, held in conjunction

with international conference on supercomputing (WSSPE)
Qiu J, Wang J, Yao S, Guo K, Li B, Zhou E, Yu J, Tang T, Xu N, Song S et al (2016) Going deeper with embedded FPGA platform for convolutional neural network. In: Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays. ACM, pp 26–35
Roy A, Bindschaedler L, Malicevic J, Zwaenepoel W (2015) Chaos: scale-out graph processing from secondary storage. In: Proceedings of the 25th symposium on operating systems principles. ACM, pp 410–424
Salihoglu S, Widom J (2013) GPS: a graph processing system. In: Proceedings of the 25th international conference on scientific and statistical database management. ACM, p 22
Schelter S, Satuluri V, Zadeh R (2014) Factorbird-a parameter server approach to distributed matrix factorization. arXiv preprint arXiv:1411.0602
Shun J, Blelloch GE (2013) Ligra: a lightweight graph processing framework for shared memory. In: ACM SIGPLAN notices, vol 48(8). ACM, pp 135–146
Shun J, Dhulipala L, Blelloch GE (2015) Smaller and faster: parallel processing of compressed graphs with ligra+. In: Data compression conference (DCC). IEEE, pp 403–412
Wang W, Chen G, Dinh ATT, Gao J, Ooi BC, Tan KL, Wang S (2015) SINGA: putting deep learning in the hands of multimedia users. In: Proceedings of the 23rd ACM international conference on multimedia. ACM, pp 25–34
Wang Y, Davidson A, Pan Y, Wu Y, Riffel A, Owens JD (2016) Gunrock: a high-performance graph processing library on the GPU. In: ACM SIGPLAN notices 51(8). ACM, p 11
White T (2012) Hadoop: the definitive guide. O'Reilly Media, Inc., Sebastopol
Xing EP, Ho Q, Dai W, Kim JK, Wei J, Lee S, Zheng X, Xie P, Kumar A, Yu Y (2015) Petuum: a new platform for distributed machine learning on big data. IEEE Trans Big Data 1(2):49–67
Yan D, Cheng J, Xing K, Lu Y, Ng W, Bu Y (2014) Pregel algorithms for graph connectivity problems with performance guarantees. Proc VLDB Endow 7(14):1821–1832
Yan D, Huang Y, Liu M, Chen H, Cheng J, Wu H, Zhang C (2017) Graphd: distributed vertex-centric graph processing beyond the memory limit. IEEE Trans Parallel Distrib Syst 29(1):99–114
Yang F, Li J, Cheng J (2016) Husky: towards a more efficient and expressive distributed computing framework. Proc VLDB Endow 9(5):420–431
Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on networked systems design and implementation. USENIX Association
Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles. ACM, pp 423–438
Zhang C, Li P, Sun G, Guan Y, Xiao B, Cong J (2015) Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays. ACM, pp 161–170
Zhao Y, Yoshigoe K, Bian J, Xie M, Xue Z, Feng Y (2016) A distributed graph-parallel computing system with lightweight communication overhead. IEEE Trans Big Data 2(3):204–218
Zhong J, He B (2014) Medusa: simplified graph processing on GPUs. IEEE Trans Parallel Distrib Syst 25(6):1543–1552
Zhou C, Gao J, Sun B, Yu JX (2014) Mocgraph: scalable distributed graph processing using message online computing. Proc VLDB Endow 8(4):377–388

Scalable Big Data Privacy with MapReduce

Sibghat Ullah Bazai[1], Julian Jang-Jaccard[1], and Xuyun Zhang[2]
[1]Institute of Natural and Mathematical Sciences, Massey University, Auckland, New Zealand
[2]Department of Electrical and Computer Engineering, University of Auckland, Auckland, New Zealand

Overview

Processing big data to drive useful information has been in spotlight in recent years. Numerous approaches have been proposed to explore different ways to analyse the big data. However, data privacy has been an issue during the process because data could have been from various sources and they may contain sensitive personal information of individual. Hadoop MapReduce has been considered as one of the most promising approaches for big data processing. This chapter provides an overview of MapReduce environment, privacy challenges faced during the processing of data in MapReduce cluster, existing approaches adopted by various researchers to mitigate these issues. We also provide future guidelines for anonymized data processing to ensure individual privacy in MapReduce.

Introduction

Big data analytics is an emerging technology for finding new insights from large amounts of data. Processing and analyzing these large amounts of data require an extra set of tools and services. Traditional approaches for processing such data unfortunately are no longer effective because of its inherent limitations like computation power, storage capacity, analyzing, visualizing, searching, and sharing (Adnan et al. 2014). For example, Google searching is accessed 38,000 times in 1 s, more than 2.2 billion users send 144 billion emails in 1 day, Facebook manages its users' 40 billion photos, and Walmart stores approximately 2.5 petabytes of data and manages more than million customer transactions per hour (Patel et al. 2012). In the past years, the requirement for processing and storing data used in traditional approaches was quite different than they are today. For example, the traditional approaches did not provide scalability to complete the processing requirement within a given time (Peralta et al. 2015). Scalability is essential when the dataset is too large to be processed on a single machine or the processing becomes unacceptably slow. A careful plan was needed ahead of time to avoid unnecessary hassles, for example, processing the data in a multiple nodes. Similarly, some tasks are simply too time consuming as often computation required to process the data is too complex and resource hungry to run on a single computer. Running machine learning algorithms on large datasets is one of such tasks.

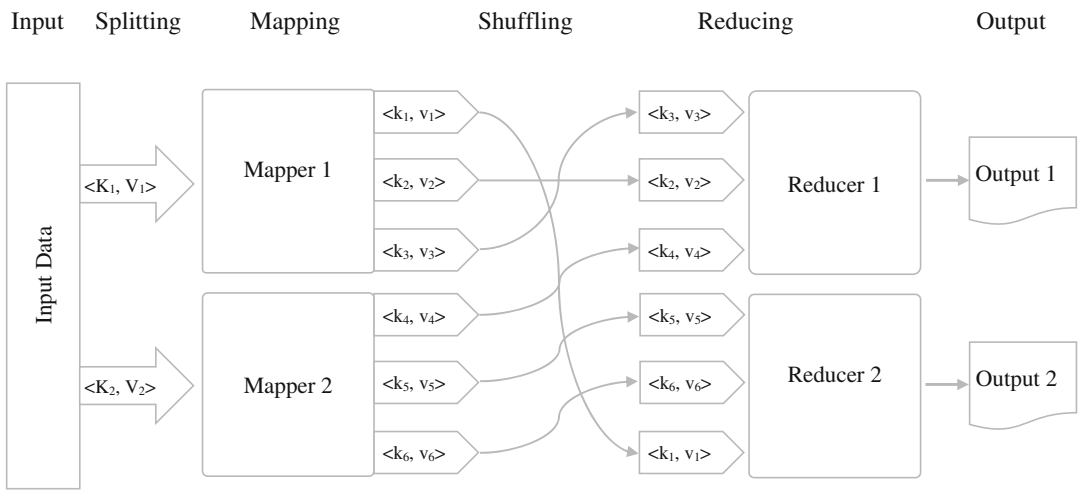
Fortunately, the commodity hardware has become cheaper, and putting several computers together to provide a high processing power has become easier in many cases. Even more, many third-party companies, such as Cloud, offer an hourly rental of hardware/software resources which has made the horizontal scaling very attractive lately. Many frameworks have been designed to use on the commodity hardware/software for specific computing which include MapReduce and Apache Spark among other choices (Dean and Ghemawat 2008; Zaharia et al. 2010).

Data owners are continuing in search of multi-node cluster platform which provides a powerful parallel and distributed architecture to process the big data fast and more efficiently (Aggarwal and Philip 2008). However, the privacy of data processed in such platform has given a data mining community a new concern. There are various techniques to achieve the privacy which includes perturbation, cryptographic-based techniques such as homomorphic encryption and secure multiparty computation, and data anonymization-based techniques such as k-anonymity and differential privacy. However, these techniques exist in isolation from each other in which often are tailored to address a specific problem of a specific domain. This chapter will highlight current big data platform architecture through MapReduce programming model, privacy issues found while processing big data in such architecture, and how existing approaches can (or can't) mitigate the privacy concern. Finally, this chapter will conclude with the future research directions which can provide more advanced approaches to address current privacy concerns.

What Is MapReduce?

The MapReduce-based solutions are considered to be better suited for big data analytics by some as it provides both flexibility and scalability. The MapReduce framework consists of two main functions, mapper and reducer. The mapper function is responsible for splitting an input job (typically large) into a number of smaller manageable jobs. The reducer function collects the divided smaller jobs and processes it further then writes the result to an output file. Figure 1 shows a diagram for the MapReduce programming model.

The mapper function is responsible for splitting an input file into smaller jobs where each job is defined by a key (i.e., an id to a job) and value (i.e., actual job) pairs. The map phase consists of multiple mappers – the same map function is carried out on each mapper. Mappers are executed independently, meaning that there are no interactions among them. There is no



Scalable Big Data Privacy with MapReduce, Fig. 1 The MapReduce programming model

relationship between the input and output types. Figure 1 shows the input to the mapper phase which is abstract as $< k_i, V_j >$ and the output of the map phase which is abstract as $< k_x, v_y >$. The mapper functions produce partial results, and these partial results are shuffled and sorted by respective key once the mapper function execution is completed. An intermediate data is stored in a local disk which contain the results of shuffled and sorted key value pairs.

Reducer functions collect the intermediate data and perform aggregation on it. Once the mapper tasks are complete, each reducer function accepts a list of values for a same key. The inputs for the reducer function are from the intermediate data. For the list of values, the programmer can define their own process. Finally, the final outputs of the reduce function are emitted at last. The output is still in tuple format (i.e., key and their values) where the key remains the same as input key, while the value is usually the aggregation of all the values in the input. Output tuples are written to their respective files in the file system.

Mapper and reducer functions execute their respective parts separately on different nodes which are often distributed and run in parallel. Hadoop (White 2012) implements MapReduce with Hadoop Distributed File System (HDFS) for an open-source project. This project automatically handles task distribution, fault tolerance, and other aspects of distributed computing. This makes it much easier for programmers to write parallel processing. It also enables Google to develop a large number of commodity computers to achieve high performance computing at a fraction of the cost compared to a system that is built from a fewer but more expensive high-end servers. MapReduce scales performance by scheduling parallel tasks on several nodes that store the task inputs. Each node executes the tasks with loose communication with other nodes. HDFS is an open-source DFS inspired by Google File System (GFS) and designed to run Hadoop's batch jobs in massive parallelism. HDFS does not implement leases and users can choose which data replica is to be written. HDFS does data-balancing during writes and not periodically like GFS.

Privacy Concern

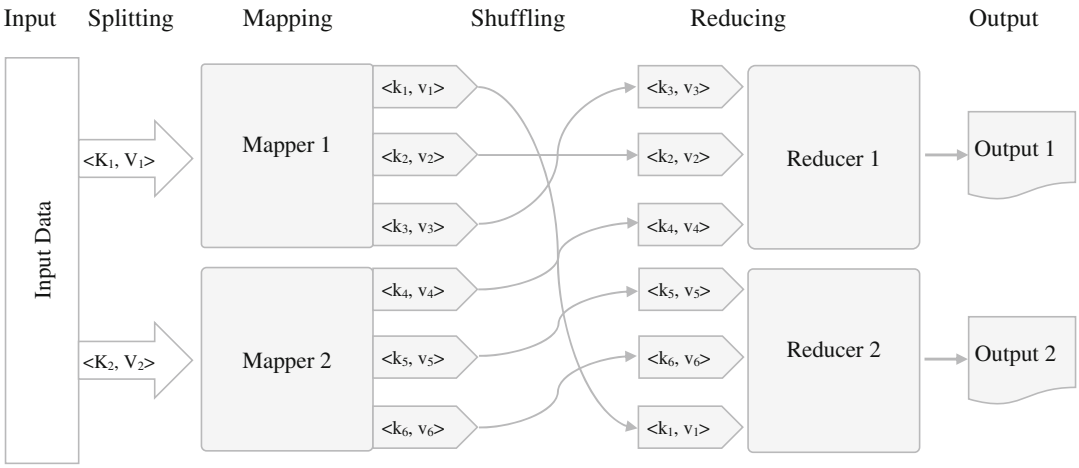
Unfortunately, distributed data and replicated nature in the MapReduce processing opens up an opportunity for a new huge variety of threats and attacks. While these threats share similarities for attacks in cloud computing models, the effects of attacks for the MapReduce framework are

quite different (Derbeko et al. 2016). MapReduce deployed in public clouds is more open to threats and attacks compare when it is deployed in private clouds. The private cloud deployment is more secure because it is deployed in a trusted boundary of a company. This can reduce the danger of attacks and can also prevent company's resources from the threats. Executing the MapReduce jobs in public clouds introduces new challenges in the form of data privacy. In MapReduce, there are multiple mappers and reducers involved in processing data. If one of the many mappers or reducers is compromised by an adversary, it may provide incorrect outputs. A powerful adversary can compromise mapper and reducer function to access information from input, intermediate process, or final output. In these situations, it becomes increasingly difficult to protect individual's privacy in mapper or reducer function. In addition, in MapReduce operations, mappers transform input original key and value pairs to calculate an intermediate key and value pairs, while reducers aggregate an intermediate values set, compute, and write to the output. The output however can bring serious privacy concerns. Firstly, the output can directly leak sensitive information because it contains the global view of the final computation. Secondly, the output can also indirectly leak information via composite attacks where the adversary can link it with public information published via different sources such as Facebook or Twitter. Due to these reasons cloud users hesitate to upload their private information in public clouds. A Cloud computing client may delete sensitive information of individuals present in dataset to ensure their privacy and process the dataset in the public clouds because of its size and convince of computing in the said cloud platform. A Cloud user can delete their sensitive information if the user thinks their data is at risk. However, this is still not enough to protect the user data privacy because third-party administrator controls the public cloud. Third party administrator can easily monitor and eavesdrop client information, and the MapReduce code. Ensuring privacy protection in a compromised cloud service provider is a big concern and can breach the primary aim of privacy which is to allow clients to use MapReduce services without compromising their data privacy.

Critiques on Emerging Approaches

The conventional anonymization approaches were applied on the output of MapReduce to provide privacy protection. Although it ensures the privacy of data but if the adversary has any access to MapReduce, processing it may leak private information before anonymization is applied to the data. The MapReduce processes data in plain text which is venerable from unauthorized monitoring and modification if data operations are executed in untrusted boundaries. In recent years, researchers have adopted anonymization approaches not only on the output but in different stages of MapReduce platform to ensure privacy. In Fig. 2, the areas highlighted in gray are used for anonymization. More number of research is done in the reducer function like Airavat (Roy et al. 2010), Sedic (Zhang et al. 2011), and others. However, the some studies also focus on mapper function to ensure privacy before exaction in plain text and explained as follows.

The current MapReduce framework cannot fully fulfill the requirement for privacy and security features for various applications and infrastructures. One of the cryptographic approaches for privacy-preserving computations is the homomorphic encryption. The technique, considered to be the holy grail of cryptography (Micciancio 2010), allows to perform computation on the cipher text itself without requiring any decryption. The usefulness of such a scheme is evident in the era of cloud computing, particularly in scenarios where organizations are worried about the privacy of their data contents. In traditional cryptographic schemes, the actual data, called plaintext, is encrypted to a distorted representation, called the ciphertext, which is not suitable for direct computations. In such schemes, we must decrypt the ciphertext to the actual data so that we can perform meaningful computations. This, however, is not the case with homomorphic



Scalable Big Data Privacy with MapReduce, Fig. 2 Anonymised MapReduce programming model

encryption schemes where, even after converting to the ciphertext, we can perform certain evaluations without seeing the plaintext. In this way, the plaintext can be kept secret from a third party who can perform calculations on the ciphertext and still provide the correct results. Another most common approach for protecting the security and privacy of data would be to use secure multiparty protocols for computation. This approach has long been studied as in Goldreich et al. (1987). In this technique, a protocol is established between the parties involved such that they all participate in calculating a desired result but without knowing the inputs of other parties (Goldreich 1998). The major cryptographic techniques used in MPC are homomorphic encryption (Cramer et al. 2001) and garbled circuits (Yao 1982). Kamal et al. (Dankar and El Emam 2012) introduce a cryptographic cloud storage service that enables search operation on encrypted data in the cloud, while also ensuring that no information is leaked in the process, and also allows users to verify the integrity of the data at any time. The proposed solution utilizes the combination of attribute-based encryption, searchable encryption, and proof of storage. Mayberry et al. proposed a private information retrieval for MapReduce (PIRMAP). PIRMAP uses mapper phase for parallel computation while homomorphic aggregation is performed in reducer function (Mayberry et al. 2013). PRISM, a MapReduce technique introduced by Blass et al. (2012), can be efficiently computed in MapReduce. However, these approaches do provide privacy during the execution process but also significantly increase time to encrypt and decrypt the data during computation and do not support many operations on encrypted data.

HybrEx (Ko et al. 2011) and Sedic (Zhang et al. 2011) use MapReduce framework in hybrid cloud to provide privacy for computation data. The HybrEx model presents four execution models for providing privacy to data in a hybrid cloud, that is, map hybrid, horizontal partitioning, vertical partitioning, and hybrid. Each execution model is designed for different setting of map, reducing function execution in multiparty cloud environment. However, the paper does not provide any implementation for any of the proposed model. Sedic implemented the map hybrid model on top of Hadoop, in which reducer operation is only executing in the private cloud without utilizing the public cloud resources. However, the sensitivity of given dataset should be defined in the system before the execution, which eliminates the chances of using chained or iterative MapReduce process. Roy et al. (2010) present Airavat, a security and privacy preservation platform for MapReduce. Airavat ensures individual's data privacy using differential privacy by

adding calculated noise to each query in MapReduce. There are two approaches proposed by the authors. First, it provides mandatory access control (MAC) to ensure only authorized users have access to authorized tasks and resources. The MAC is activated when privacy leakage exceeds a defined limit. Tran and Sato (2012), however, if adversary, manage to sneak reducer code by changing user rights as a trusted user; the proposed solution fails to provide privacy guarantee. Zhang et al. (2012) proposed a privacy-preserving layer over MapReduce, which satisfies privacy demands itemized by data publishers built on diverse MapReduce privacy models. Mohan et al. then present GUPT (Mohan et al. 2012) which is another improved version of Airavat. GUPT defines the differentially private parameters in such a way that it automatically chooses and distributes the privacy parameter. However, Airavat add p reconfigured noise for query which limits its application. Clifton and Tassa (2013) observed that Airavat, unable to pick a suitable privacy budget and sensitivity value for selective quarry, leaves the privacy choice up to the user. In addition, Xiao and Xiao (2014) assign the executing task to new worker node and compare both results to eliminate the malicious node. The existing schemes have certain limitations such as inefficiency to handle incremental data, time taken to update records is more, experiences poor scalability, absence of parallelization, poor execution time, higher rate for loss of information, and inequity between data utility and anonymization.

Future Direction

Processing big data for multi-node cluster involved many vulnerabilities, for example, administrative control of data processing environment, distribution of data on an untrusted node in a public cloud for processing, etc. Victor et al. (2016). To address this limitation, state-of-the-art privacy protection methodologies, such as k-anonymity and differential privacy, have received great attention from the scientific community in recent years, and there are few implementations offered in the area (Bello-Orgaz et al. 2016). However, these approaches were adopted in the earlier versions of big data processing platform like earlier version of Hadoop. Since the architecture and design for these approaches have been improved radically, the data anonymization algorithm used for previous deployments also needs to be updated to work efficiently (Jain et al. 2016).

K-anonymity ensures the privacy of data where perturbation and cryptographic approaches failed with well-known datasets such as AOL, adult dataset, and Netflix (Bayardo and Agrawal 2005). K-anonymity is the first data anonymization technique with formal mathematical support as a proof. Latanya Sweeney in 2002 (Sweeney 2002) introduced k-anonymity by stating that without ensuring k individuals in aggregation single aggregate statistic should not be published. The formal definition of k-anonymity formalized as each release of the data must be such that every combination of values of quasi-identifiers can be indistinguishably matched to at least k respondents. Quasi-identifiers (QID) are attributes in dataset which may be linked from publicly available dataset by join linkage. The main goal to achieve k-anonymity is to replace QID values with more general values, e.g., generalizing 15, 17, and 19 into a 15–20 value. K-anonymity provides the balance between privacy and utility. It provides more flexibility to data publisher for choosing the desired level of privacy. Most importantly, the anonymized data can be used for processing for analytical data operations even after applying data anonymization technique. Bazai et al. (2017a) proposed an algorithm to use k-anonymity on k-NN classifier for MapReduce operations. The algorithm transformed the plaintext data into anonymized data and executed k-NN classifier on anonymized data. The results provide better data utility for veracious degrees of anonymity. Differential privacy approach has been widely used in statistical queries (Inan et al. 2010). Differential privacy ensures the privacy of individual by answering the query in such a way that the query result does not contain any information about individual's presence in

that data. Differential privacy (Dwork 2008) was formally presented by Dwork in 2006, the idea of adding carefully calibrated noise to the data. The noise level is controlled in such a way that if the records are correlated with other datasets, then an adversary is not able to trace the data subject. The addition of noise in the original data is calibrated so that the noisy data acquired must be usable for data processing in the same way as the original. The noise is not allowed to increase beyond a level where the noisy data becomes so different from the original data that its utilization (accuracy) is impractical. Differential privacy has many advantages over other privacy techniques. Firstly, the mathematical definition of differential privacy is based on two properties: sensitivity and privacy budget. The sensitivity focuses on record closeness to each other, and the privacy budget is calculated based on how much privacy is required for a given query. Secondly, also only unique to this technique, it does concern with adversary computational power and background knowledge (i.e., even if the adversary knows the individual's publicly available details and then uses it for de-anonymization of the query, the adversary cannot de-anonymize it because of random anonymization mechanism used by differential privacy). Thirdly, it maintains composability which ensures that it does not leak privacy even an adversary holds and combines two differentially private results. These properties free data providers from the concern of privacy leakage of their data. Vernica et al. describe KNN processing approach for computing set resemblances on textual documents (Vernica et al. 2010). Their focuses are with document processing rather than business data analytics, and they do not address any implication of privacy; therefore, no data anonymization strategies were mentioned.

In big data processing platforms, computation is done on data in different executions in plaintext; any information leakage while processing the data may disclose sensitive information. Bazai et al. (2017b) mainly focus on addressing this limitation and intended to explore how data anonymization techniques (k-anonymity and differential privacy) can be applied in big data processing environment, using MapReduce, which provides a well-known basic set of big data processing architecture and infrastructure, to validate the proposed approach.

Big datasets usually contain many different data types; for example, information is stored in the textural data containing categorical and continuous value, while others are stored in the numerical data. Previous studies show (Fletcher and Islam 2017) that different data types impose different constraints when applying data anonymization techniques. This is true when each privacy parameter has a different impact on the data type. For example, k-group size provides the better utility and privacy with categorical values, while differential privacy works better with well-distributed numerical values (Blum et al. 2013). Many different approaches for using different privacy parameters have been found in the literature for categorical values and numerical data in traditional data processing approaches (Natwichai et al. 2006); however, these implementations do not support multi-node cluster.

Summary

Big data is a very popular term nowadays and has opened a big data era. Distributed and parallel applications are designed to process this huge data to drive new insights and knowledge. Although the design of these applications ensures the data security in a certain level, individual privacy is at a great risk. This chapter illustrates the privacy concern of big data processing platform and provides critiques for existing approaches that are supposed to prevent MapReduce privacy leakage. Although researchers are addressing the privacy concern with various approaches for data analytics in big data processing platforms with perturbation, cryptographic techniques, k-anonymity, and differential privacy, most of the studies are still are in early stages. The practical implementation of anonymity with data analytics will help number of privacy and security challenges.

References

Adnan M, Afzal M, Aslam M, Jan R, Martinez-Enriquez A (2014) Minimizing big data problems using cloud computing based on hadoop architecture. In: 2014 11th annual high-capacity optical networks and emerging/enabling technologies (HONET). IEEE, pp 99–103

Aggarwal CC, Philip SY (2008) A general survey of privacy-preserving data mining models and algorithms. In: Privacy-preserving data mining. Springer, Dordrecht, pp 11–52

Bayardo RJ, Agrawal R (2005) Data privacy through optimal k-anonymization. In: Proceedings of the 21st international conference on data engineering (ICDE 2005). IEEE, pp 217–228

Bazai SU, Jang-Jaccard J, Wang R (2017a, in press) Anonymizing k-nn classification on mapreduce. In: The 9th EAI international conference on mobile networks and management. Springer

Bazai SU, Jang-Jaccard J, Zhang X (2017b) A privacy preserving platform for mapreduce. In: International conference on applications and techniques in information security. Springer, pp 88–99

Bello-Orgaz G, Jung JJ, Camacho D (2016) Social big data: recent achievements and new challenges. Inf Fusion 28:45–59

Blass EO, Di Pietro R, Molva R, Önen M (2012) Prism-privacy-preserving search in mapreduce. In: Privacy enhancing technologies, vol 7384. Springer, pp 180–200

Blum A, Ligett K, Roth A (2013) A learning theory approach to noninteractive database privacy. J ACM (JACM) 60(2):12

Clifton C, Tassa T (2013) On syntactic anonymity and differential privacy. In: 2013 IEEE 29th international conference on data engineering workshops (ICDEW). IEEE, pp 88–93

Cramer R, Damgård I, Nielsen J (2001) Multiparty computation from threshold homomorphic encryption. In: Advances in cryptology-EUROCRYPT 2001, pp 280–300

Dankar FK, El Emam K (2012) The application of differential privacy to health data. In: Proceedings of the 2012 joint EDBT/ICDT workshops. ACM, pp 158–166

Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

Derbeko P, Dolev S, Gudes E, Sharma S (2016) Security and privacy aspects in mapreduce on clouds: a survey. Comput Sci Rev 20:1–28

Dwork C (2008) Differential privacy: a survey of results. In: International conference on theory and applications of models of computation. Springer, pp 1–19

Fletcher S, Islam MZ (2017) Differentially private random decision forests using smooth sensitivity. Exp Syst Appl 78:16–31

Goldreich O (1998) Secure multi-party computation. Manuscript preliminary version, pp 86–97

Goldreich O, Micali S, Wigderson A (1987) How to play any mental game. In: Proceedings of the nineteenth annual ACM symposium on theory of computing. ACM, pp 218–229

Inan A, Kantarcioglu M, Ghinita G, Bertino E (2010) Private record matching using differential privacy. In: Proceedings of the 13th international conference on extending database technology. ACM, pp 123–134

Jain P, Gyanchandani M, Khare N (2016) Big data privacy: a technological perspective and review. J Big Data 3(1):25

Ko SY, Jeon K, Morales R (2011) The hybrex model for confidentiality and privacy in cloud computing. In: HotCloud, pp 1–8

Mayberry T, Blass EO, Chan AH (2013) PIRMAP: efficient private information retrieval for mapreduce. In: International conference on financial cryptography and data security. Springer, pp 371–385

Micciancio D (2010) A first glimpse of cryptography's holy grail. In: Commun ACM 53(3):96–96

Mohan P, Thakurta A, Shi E, Song D, Culler D (2012) GUPT: privacy preserving data analysis made easy. In: Proceedings of the 2012 ACM SIGMOD international conference on management of data. ACM, pp 349–360

Natwichai J, Li X, Orlowska ME (2006) A reconstruction-based algorithm for classification rules hiding. In: Proceedings of the 17th Australasian database conference, vol 49. Australian Computer Society, Inc., pp 49–58

Patel AB, Birla M, Nair U (2012) Addressing big data problem using hadoop and map reduce. In: 2012 Nirma University international conference on engineering (NUiCONE). IEEE, pp 1–5

Peralta D, del Río S, Ramírez-Gallego S, Triguero I, Benitez JM, Herrera F (2015) Evolutionary feature selection for big data classification: a mapreduce approach. In: Math Probl Eng, pp 1–12

Roy I, Setty ST, Kilzer A, Shmatikov V, Witchel E (2010) Airavat: security and privacy for mapreduce. In: NSDI, vol 10, pp 297–312

Sweeney L (2002) Achieving k-anonymity privacy protection using generalization and suppression. Int J Uncertain Fuzziness Knowl Based Syst 10(05):571–588

Tran Q, Sato H (2012) A solution for privacy protection in mapreduce. In: 2012 IEEE 36th annual computer software and applications conference (COMPSAC). IEEE, pp 515–520

Vernica R, Carey MJ, Li C (2010) Efficient parallel set-similarity joins using mapreduce. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM, pp 495–506

Victor N, Lopez D, Abawajy JH (2016) Privacy models for big data: a survey. Int J Big Data Intell 3(1):61–75

White T (2012) Hadoop: the definitive guide. O'Reilly Media, Inc., Sebastopol

Xiao Z, Xiao Y (2014) Achieving accountable mapreduce in cloud computing. Futur Gener Comput Syst 30:1–13

Yao AC (1982) Protocols for secure computations. In: 23rd annual symposium on foundations of computer science, SFCS'08. IEEE, pp 160–164

Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I (2010) Spark: cluster computing with working sets. HotCloud 10(10–10):95

Zhang K, Zhou X, Chen Y, Wang X, Ruan Y (2011) Sedic: privacy-aware data intensive computing on hybrid clouds. In: Proceedings of the 18th ACM conference on computer and communications security. ACM, pp 515–526

Zhang X, Liu C, Nepal S, Dou W, Chen J (2012) Privacy-preserving layer over mapreduce on cloud. In: 2012 second international conference on cloud and green computing (CGC). IEEE, pp 304–310

Scalable SPARQL Query Processors

▸ Native Distributed RDF Systems

Scaling up OLTP on Multicores and Many Cores

▸ Hardware-Assisted Transaction Processing

Schema Mapping

Paolo Papotti
Campus SophiaTech – Data Science
Department, EURECOM, Biot, France

Synonyms

Mapping

Definitions

Several data management tasks require to precisely establish associations between data structured under different schemas. Schema mappings allow the definition of semantic connections between schemas with structural differences. A schema mapping M is a specification of a relation between instances of a source schema S and instances of a target schema T. Given a source instance I for S and a target instance J for T, they satisfy the mapping if $(I, J) \models M$. Schema mappings are used both for virtual and materialized integration, and they have been studied in many aspects, including their specification, semantics, and user-assisted generation.

Overview

There are many applications that need to exchange, correlate, and integrate heterogeneous data sources. These information integration tasks have long been identified as important problems, and unifying theoretical frameworks have been advocated by database researchers (Bernstein and Melnik 2007). To solve these problems, a fundamental requirement is that of manipulating *mappings* among data sources. The application developer is typically given two schemas – one called the source schema S and the other called the target schema T – that can be based on radically different models and structures. *Schema mappings* are logical expressions that specify the semantic connection between the instances for the source and the target schemas. Given a source instance I and a target instance J that satisfy the schema mapping, we can say that $(I, J) \models M$.

A mapping is usually represented as a triple $< S, T, M >$, and its semantics is defined with respect to the setting where only the source instance I (for S) is given. Before introducing the semantics, it is useful to distinguish the two main problems for which mappings play a key role: *data exchange* (Fagin et al. 2005a) and *data integration* (Lenzerini 2002). The two problems share the same goal: support users in retrieving source data by posing queries on the target, which is also known as global or mediated schema in data integration. The main difference is in the way that such goal is achieved. *Data exchange* formally studies the semantics of generating an instance of a target database given a source instance, a mapping, and constraints on the target schema. It has formalized the notion of a *data exchange*

problem and has established a number of results about its properties. In data exchange, the source data is transformed into a materialized instance that conforms to the target schema. Users define queries on the target schema, and these are then answered by using the target instance. *Data integration* is the problem of combining data residing at different sources and providing the user with a unified view of these data. The main difference with data exchange is that the query is part of the input and only the answer to such query is materialized. The source data is queried in situ, i.e., the target queries are rewritten to compute answers using the source data only. Despite the differences in the execution, in both approaches the goal is to reason about the possible instances J such that $(I, J) \models M$.

The most popular logical specification for schema mappings are source-to-target tuple-generating dependencies (s-t TGDs), with the following form:

$$\forall x(\Phi_S(x) \rightarrow \exists y \Psi_T(x, y)) \tag{1}$$

where $\Phi_S(x)$ and $\Psi_T(x, y)$ are conjunctions of atomic formulas over S and T, respectively. Consider, for example, a source schema with two relations *Patient (SSN, Name, Phone, HosDate)* and *Surgery (PatName, Insurance, Treatment, Date)* and the target schema with two relations *Customer (Id, Name, Phone, BirthDate, Insurance)* and *Claim (CustId, Treatment, Date)*. A s-t TGD defined over these two schemas is

$$\begin{aligned}&\forall p, i, t, d\ Surgery(p, i, t, d) \rightarrow \\ &\quad \exists Y_1, Y_2, Y_3(Customer(Y_1, p, Y_2, Y_3, i) \wedge \\ &\quad Claim(Y_1, t, d))\end{aligned}$$

Assume the TGD above is the only constraint in our mapping M. Consider now a given instance I for the source schema, such as I = *Surgery(Mark,Axa,Eyes,2/21/17)*, and three target instances:

$$\begin{aligned}J_0 = &Customer(10, Mark, 978, 3/3/79, Axa), \\ &Claim(20, Eyes, 2/21/17)\end{aligned}$$

$$\begin{aligned}J_1 = &Customer(11, Mark, 978, 3/3/79, Axa), \\ &Claim(11, Eyes, 2/21/17)\end{aligned}$$

$$\begin{aligned}J_2 = &Customer(12, Mark, 978, 3/3/79, Axa), \\ &Claim(12, Eyes, 2/21/17)\end{aligned} \tag{2}$$

It is easy to see that the pair of instances (I, J_0) does *not* satisfy the s-t TGD specification, because of the different values for the customer id. In this case, we say that J_0 is not a *solution* for the given I and mapping M. On the other hand, both instances J_1 and J_2 satisfy the TGD together with I. Since $(I, J_1) \models M$ and $(I, J_2) \models M$, we say that each of the two target instances is a solution for I. One may observe that these two target instances have different customer ids, but this is not in contradiction with neither the source instance, which does not have this information, nor the mapping, as the s-t TGD only states that the id value should the same in the *Customer* and *Claim* relations.

The example represents a mapping that specifies what source data must be in the solutions, but does not specify what must not be in any solution. For example, a fourth target instance J_3

$$\begin{aligned}&Customer(12, Mark, 978, 3/3/79, Axa), \\ &Claim(12, Eyes, 2/21/17), \\ &Claim(12, Liver, 10/1/2010)\end{aligned} \tag{3}$$

is also a solution for I and M above. The s-t TGDs above are in fact called *open* mappings and compose the most studied class of mappings in the literature (Bernstein and Melnik 2007). These mappings always allow a solution and can be specified with no requirements on the target or other sources (i.e., other open mappings cannot conflict). However, there are more constrained settings where exact mappings have been studied (Fuxman et al. 2006b).

Another important characterization of schema mappings is with respect to their structure (Lenzerini 2002). This comes from data integration systems, which assumed that the schema mapping is a function, e.g., a view. In one setting, known as global-as-view (GAV), mappings have

one relation symbol in $\Psi_T(x, y)$ and no repeated variables. In the opposite modeling, each relation of the source schema is defined in terms of the target schema. In this setting, known as local-as-view (LAV), mappings have one relation symbol in $\Phi_S(x)$ and no repeated variables. GAV and LAV have been largely studied and adopted because of their tractable data complexity when answering queries in data integration systems.

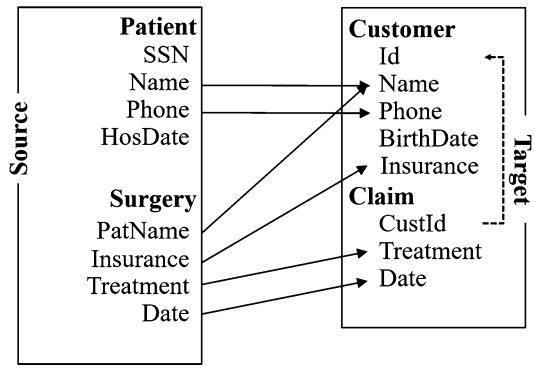
Key Research Findings

One important property of schema mappings is that they can be generated from graphical user interfaces (GUI). In fact, manually designing a schema mapping between schemas can be challenging. Even if the schemas represent similar data, they very often use different names and structures to describe the information. This is especially true when the schemas are independently designed. Manually crafting schema mappings is a difficult task, which requires technical experts that have deep understanding of both source and target schemas. When the size of the schemas increases, with dozens of relations and hundreds of attributes, it is also easy to make errors in the mappings, which can then lead to incorrect results in the ultimate application.

Since understanding the design of the schemas and manually writing schema mappings is time-consuming and prone to human errors, *mapping generation tools* have been created to make the process more abstract and user-friendly, thus easier to handle for a larger class of people. Most of these tools have been designed with data exchange in mind and directly compared to other solutions for the design of data transformation, such as ETL systems (Dessloch et al. 2008; Mecca et al. 2012). However, the same tools can be used to create schema mappings in data integration settings. Schema mapping generation tools include primarily Clio, which is the first tool able to generate logical mappings and corresponding scripts from high-level user input (Fuxman et al. 2006a; Haas et al. 2005; Miller et al. 2000; Popa et al. 2002). Three main features characterize mapping generation tools.

High-level interface. The goal of simplifying the mapping specification was pursued by introducing a GUI that allows users to draw arrows, or *correspondences*, between schemas in order to define the desired transformation. Such correspondences can also be discovered and suggested to the users by schema matching algorithms (Bernstein et al. 2011). Correspondences represent associations between elements across different schemas in the form of attribute pairs and are discovered by profiling the attribute values and by mining the schema structures.

Consider the example shown in Fig. 1, with the relations described earlier and a foreign key from *Claim* to *Customer* in the target schema. Correspondences, represented as solid arrows in the figure, map atomic elements of the source schema to elements of the target schema, independently of the underlying data model or of the design choices. Due to the heterogeneity of the databases, the same concept could be represented differently across different schemas. At the same time, portions of the schemas with similar structure may actually model different entities. In the figure, the customer name in the target is encoded as the name of a patient in the source. However, multiple date attributes are present in both schemas, but they have different semantics across the relations, e.g., hospitalization date is



Schema Mapping, Fig. 1 Schema mapping scenario

different from birth year. While correspondences are easy to create and understand, they do not represent the full semantic relationship between source and target instances. In the example, the correspondences from the source relation *Surgery* to the target relations specify the constraint over the names and the treatments, but do not specify the relationship between the instances in relation *Customer* and *Claim*. This relationship is expressed explicitly by the foreign key and is part of the semantic specification, but it is lost in the attribute to attribute granularity of the correspondences. As we have seen in the schema mapping (2), a more complex formula is needed to capture precisely the semantic associations at the schema level. Correspondences are therefore taken as input by a schema mapping tool.

Automatic Mapping Generation. Based on value correspondences, mapping systems generate the logical dependencies that specify the mapping. In an operational interpretation, the resulting TGDs can be seen as a specification of how to *translate* data from the source to the target. It has been shown that the mapping generation is sound and complete with respect to the given correspondences, that is, all the relevant schema mappings are part of the output and none of them contradicts the correspondences (Fuxman et al. 2006a; Popa et al. 2002). Based on the correspondences drawn in Fig. 1, a mapping system would generate the following schema mapping with two dependencies:

SOURCE-TO-TARGET TGDS

$$
\begin{aligned}
&m_1.\forall s,n,p,h\ \ Patient(s,n,p,h) \rightarrow \\
&\exists Y_1,Y_2,Y_3\,(Customer(Y_1,n,p,Y_2,Y_3)) \\
&m_2.\forall p,i,t,d\ \ Surgery(p,i,t,d) \rightarrow \\
&\exists Y_1,Y_2,Y_3\,(Claim(Y_1,p,Y_2,Y_3,i) \wedge \\
&Account(Y_1,t,d))
\end{aligned} \tag{4}
$$

Mapping Execution via Scripts. Given a source instance and a mapping, to produce a target instance that is a solution, it suffices to execute the traditional *chase procedure* (Fagin et al. 2005a). The chase is a fixpoint algorithm which tests and enforces implication of data dependencies, such as TGDs, in a database. After the mappings had been generated, a schema mapping system translates the s-t TGDs under the form of an SQL script. When executed on any DBMS, the script implements the chase, i.e., it can be applied to a source instance I to create a new instance J that is a solution. Given a TGD, in order to chase it over I, we may see its $\Phi(x)$ as a first-order query Q_ϕ with free variables x over the source database. We execute $Q_\phi(I)$ using SQL in order to find all vectors of constants that satisfy the premise, and we then insert the appropriate tuple into the target instance to satisfy $\Psi(x, y)$. Skolem functions (Popa et al. 2002) are typically used to automatically generate new values for the existential variables y.

Given mappings written by users or generated from tools, a number of approaches have been proposed to *optimize* schema mappings in order to improve the efficiency of their execution and manipulation in real-world applications. In fact, schema mappings may present redundancy in their expressions, due, for example, to the presence of unnecessary atoms or unrelated variables, thus negatively affecting the data management process at hand (Fagin et al. 2008). The following efforts have aimed at optimizing such dependencies by identifying two kinds of equivalence aside from standard logical equivalence: the relaxed notions of data exchange (DE) equivalence and conjunctive query (CQ) equivalence. DE and CQ equivalences coincide with logical equivalence when the mapping scenario is made only of s-t TGDs, but differ on richer classes of equivalences, such as second-order TGDs, and scenarios with target constraints. Mapping rewriting has also been studied to compute a different class of data exchange solutions that reduces the redundancy in the target instances. Given a mapping scenario composed of s-t TGDs, these algorithms rewrite them to generate a runtime script that, on input instances, materializes compact solutions at scale (Mecca et al. 2009; Ten Cate et al. 2009). These works exploit the use of *negation*

in the premise of the s-t TGDs to rewrite them intercepting possible redundancy.

Schema mapping tools have proven to be effective in easing the burden of manually specifying TGDs and have been successfully transferred into commercial (Haas et al. 2005) and open-source systems (Marnette et al. 2011). Other commercial tools that generate some form of schema mappings from correspondences include Stylus Studio and Microsoft's BizTalk Mapper and internal mapping modules in ETL systems. Given the large number of systems that have been proposed to create schema mappings, benchmarks for their systematic evaluation have been proposed (Alexe et al. 2008; Arocena et al. 2016).

Examples of Application

Schema mappings have been successfully adopted for data exchange and for data integration. They are popular because of their trade-off between expressiveness – as they can handle several real-world scenarios – and performance; formal results can be guaranteed with execution times that nicely scale over the size of the instances. However, many data management problems involve not only the design and execution of schema mappings but also their subsequent manipulation in a general framework (Bernstein and Melnik 2007). In this framework, other database applications, such as object-to-relational mappings and data warehousing, can be handled. This is achieved by introducing high-level algebraic operators that enable to build programs that directly manipulate schema mappings. One important application is *schema evolution*, where two of these operators play a key role. Consider a mapping M between schemas S and T, and assume that schema S evolves into a schema S' because of changes in the data organization or for performance reasons. The evolution can be expressed as a mapping M' from S to S'. Therefore, the relationship between the new schema S' and schema T can be obtained by first *inverting* the mapping M' and then *composing* the result with the original mapping M. In this scenario inversion and composition are operators that take mappings as input and return new mappings. This high-level view of schema mappings exploits the intuition that they can be the building blocks for systems that save considerable effort to the final users.

Following this vision, the composition operator has been identified as one of the fundamental operators for the development of a framework for managing schema mappings, and its semantics has been studied (Fagin et al. 2005b). The main results show that first-order (FO) logic is not sufficient to express composition and that a second-order (SO) fragment of logic can be used as a mapping language for expressing this operator. Interestingly, SO TGDs can still be executed by means of scripts, and FO schema mappings can always be translated into this more general language, thus enabling composition. There are several other operators that have been defined for schema mappings, including several proposals on how to compute the inverse of a mapping (Arenas et al. 2009) and how to merge multiple mappings with the same target schema (Alexe et al. 2010).

Future Directions of Research

Emerging trends are encouraging the developing of more systems based on schema mappings. On one side, theoretical results are paving the way to the creation of innovative applications for which schema mappings show potential positive impact, such as data pipelines and graph data (Francis and Libkin 2017; Kolaitis et al. 2016). On the other side, new algorithms for the creation and optimization of schema mappings are widening the opportunities offered by such technology, with contributions such as mapping inference from data examples and probabilistic approaches (Ten Cate et al. 2017; Kimmig et al. 2017). These recent results show opportunities for enlarging the application of schema mappings to more data management tasks, including data fusion, data cleaning, and ETL scenarios.

Cross-References

▸ Data Integration

References

Alexe B, Tan W, Velegrakis Y (2008) Comparing and evaluating mapping systems with STBenchmark. PVLDB 1(2):1468–1471

Alexe B, Hernández MA, Popa L, Tan WC (2010) MapMerge: correlating independent schema mappings. PVLDB 3(1):81–92

Arenas M, Pérez J, Reutter JL, Riveros C (2009) Composition and inversion of schema mappings. SIGMOD Record 38(3):17–28

Arocena PC, Glavic B, Mecca G, Miller RJ, Papotti P, Santoro D (2016) Benchmarking data curation systems. IEEE Data Eng Bull 39(2):47–62

Bernstein PA, Melnik S (2007) Model management 2.0: manipulating richer mappings. In: SIGMOD, pp 1–12

Bernstein PA, Madhavan J, Rahm E (2011) Generic schema matching, ten years later. PVLDB 4(11): 695–701

Dessloch S, Hernandez MA, Wisnesky R, Radwan A, Zhou J (2008) Orchid: integrating schema mapping and ETL. In: ICDE, pp 1307–1316

Fagin R, Kolaitis P, Miller R, Popa L (2005a) Data exchange: semantics and query answering. TCS 336(1):89–124

Fagin R, Kolaitis P, Popa L, Tan W (2005b) Composing schema mappings: second-order dependencies to the rescue. ACM TODS 30(4): 994–1055

Fagin R, Kolaitis P, Nash A, Popa L (2008) Towards a theory of schema-mapping optimization. In: ACM PODS, pp 33–42

Francis N, Libkin L (2017) Schema mappings for data graphs. In: Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems. ACM, PODS '17, pp 389–401

Fuxman A, Hernández MA, Howard CT, Miller RJ, Papotti P, Popa L (2006a) Nested mappings: schema mapping reloaded. In: VLDB, pp 67–78

Fuxman A, Kolaitis PG, Miller RJ, Tan WC (2006b) Peer data exchange. ACM Trans Database Syst 31(4): 1454–1498

Haas LM, Hernández MA, Ho H, Popa L, Roth M (2005) Clio grows up: from research prototype to industrial tool. In: SIGMOD, pp 805–810

Kimmig A, Memory A, Miller RJ, Getoor L (2017) A collective, probabilistic approach to schema mapping. In: 2017 IEEE 33rd international conference on data engineering (ICDE), pp 921–932

Kolaitis PG, Pichler R, Sallinger E, Savenkov V (2016) Limits of schema mappings. In: 19th international conference on database theory (ICDT 2016), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Leibniz International proceedings in informatics (LIPIcs), vol 48, pp 19:1–19:17

Lenzerini M (2002) Data integration: a theoretical perspective. In: ACM PODS, pp 233–246

Marnette B, Mecca G, Papotti P, Raunich S, Santoro D (2011) ++SPICY: an opensource tool for second-generation schema mapping and data exchange. PVLDB 4(11):1438–1441

Mecca G, Papotti P, Raunich S (2009) Core schema mappings. In: SIGMOD, pp 655–668

Mecca G, Papotti P, Raunich S, Santoro D (2012) What is the IQ of your data transformation system? In: CIKM, pp 872–881

Miller RJ, Haas LM, Hernandez MA (2000) Schema mapping as query discovery. In: VLDB, pp 77–99

Popa L, Velegrakis Y, Miller RJ, Hernandez MA, Fagin R (2002) Translating web data. In: VLDB, pp 598–609

Ten Cate B, Chiticariu L, Kolaitis P, Tan WC (2009) Laconic schema mappings: computing core universal solutions by means of SQL queries. PVLDB 2(1): 1006–1017

Ten Cate B, Kolaitis PG, Qian K, Tan W (2017) Approximation algorithms for schema-mapping discovery from data examples. ACM Trans Database Syst 42(2):12: 1–12:41

SciDB

Philippe Cudre-Mauroux
eXascale Infolab, University of Fribourg, Fribourg, Switzerland

Definitions

SciDB is a distributed database management system for managing and processing multidimensional arrays in scientific applications. It was first designed and developed by a group of academics led by Michael Stonebraker before being productized by Paradigm4.

Overview

The first XLDB workshop (1st Extremely Large Databases Workshop 2007) in 2007 brought together a group of scientists and industry members to discuss the capabilities of database management systems (DBMSs) at managing non-relational data at extreme scales. A number of

key shortcoming were identified, which were presented the following year at XLDB-2 (2nd Extremely Large Databases Workshop 2008). The lack of support for managing scientific data was in particular discussed. Consensus emerged that many Big Science projects presented unique challenges that could not be handled through generic DBMSs and would require a complete rewrite approach (Stonebraker et al. 2007).

As a result, a group of scientists led by Michael Stonebraker embarked on a new project to design and develop a new system for handling scientific arrays, as arrays represented a prominent data type for many science users (including astronomers, oceanographers, seismologists, or climate researchers). The resulting system, SciDB, was first presented at VLDB 2009 (Cudré-Mauroux et al. 2009). This first version supported a number of important features for science users, including:

1. native storage for nested, multidimensional arrays;
2. scientific operators built as user-defined functions (UDFs) to manipulate both the structure and the values of the arrays;
3. a shared-nothing design (Stonebraker 1986) allowing the system to scale out to many nodes and petabytes of data;
4. initial support for advanced features required by science users such as data versioning, provenance, and uncertainty.

The team continued the development of SciDB as an open source data management solution for Big Science projects. In 2011, Michael Stonebraker and Marilyn Matz co-founded Paradigm4 to further support the development of SciDB.

Architecture

SciDB is at its core a distributed array management system. The system takes as input large, multidimensional arrays potentially considering several values for each cell in the array. SciDB natively stores such arrays as a collection of *chunks*, each handling a portion of an array (e.g., a chunk or $10 \times 10 \times 5$ k array cells). Chunks are typically of equal size (e.g., 64 MB) and store attribute values vertically (i.e., a chunk only stores the values of a given attribute).

Chunks partition the array spatially but can be overlapping to ease some operations like object detection, which would otherwise often involve stitching together multiple chunks. Chunks are written using various compression mechanisms on disk. The system adopts a *no-overwrite* storage strategy meaning that arrays cannot be updated in place (they can however be appended or updated by creating a new version of the array). A system catalog keeps track of the attributes and the spatial position of each chunk.

SciDB distributes the chunks among a set of worker nodes. Queries consist of a tree of operators over one or multiple arrays. Operators can modify both the structure of an array (e.g., its size or dimensions) as well as its contents (attribute values). The system comes with a number of standard operators such as *Filter*, which filters cell values based on a threshold, or *SJoin*, which combines attributes from different cells. The system is extensible and allows the definition of both user-defined types (UDTs) and user-defined functions (UDFs).

Many queries, such as *filter* or *object detection*, can be processed fully in parallel on each worker node. Specific operations might however require to exchange, share, or redistribute the chunks dynamically. SciDB defines a dedicated operator, called *ScatterGather*, to help support such cases. *Scatter/Gather* is a powerful operator allowing to dynamically redistribute the array over the nodes. Queries execution is orchestrated by a central node called the *coordinator*.

Academic Prototype

The first prototype of SciDB was developed as an open source project by a group of researchers from MIT, Brow University, SLAC, NIISI RAS, the University of Washington, Portland State University, and Microsoft. The group was led by Michael Stonebraker. The system was first demonstrated at VLDB in 2009 (Cudré-Mauroux et al. 2009).

This academic prototype adopted the architecture described above, with a non-overwrite, native array storage system and distributed query execution. It also featured initial support for array versioning, data provenance and uncertainty, as well as a simple query optimizer.

The main use case for this prototype was loosely modeled on an astronomy workload and was later extended to a full benchmark for scientific data management systems (Cudre-Mauroux et al. 2010). The use case considered the end-to-end ingestion and processing of raw data from a sensor system. It included three types of operations: (i) manipulation of raw imagery, including processing pixels to extract geo-spatial observations; (ii) manipulation of observations, including spatial aggregation and grouping into related sets; and (iii) manipulation of groups, including a number of relatively complex geometric operations in several dimensions.

The academic prototype was the basis of several research projects in array data management, including:

- A new storage manager to efficiently encode and access versioned arrays (Seering et al. 2012);
- A new storage manager for complex, parallel array processing implementing various partitioning and query execution strategies (Soroush et al. 2011);
- A hybrid analytic system for array-structured data integrating R and SciDB (Leyshock et al. 2013);
- A new structure to store and model multi-dimensional arrays supporting efficient inference processes (Ge and Zdonik 2010);
- Techniques to handle fine-grained lineage in scientific databases (Wu et al. 2013);
- Techniques to incrementally add nodes and to optimize data placement for n-dimensional array systems (Duggan and Stonebraker 2014).

Paradigm4 Development

In 2011, Michael Stonebraker and Marilyn Matz co-founded Paradigm4 to further support the development of SciDB (Paradigm4 2011). The company hired Paul Brown to oversee the development of the system.

The system developed by Paradigm4 follows the architecture described above but adds a number of key features and libraries to SciDB (Stonebraker et al. 2011, 2013). It supports both a SQL-like, declarative language called *AQL* and a functional language (AFL). The system includes a full-fledged optimizer and an executor able to distribute queries to thousands of nodes. Support was also added for handling variable-size chunks, various encoding schemes (e.g., delta encoding, run-length encoding, or LZ encoding), uncertain data, and coarse-grained provenance.

A number of optimized operators come built-in with the system, including common linear algebra operators. The system also adds a number of dedicated operators for several vertical domain, e.g., for life sciences (to support genomic analysis, digital biomarker discovery, or biomedical images) and finance (for multifactor model generation or transaction cost analysis). Further use cases that have been explored include sensor analytics, insurance, and E-commerce.

Cross-References

▸ Architectures

References

1st Extremely Large Databases Workshop (2007) http://www-conf.slac.stanford.edu/xldb2007/

2nd Extremely Large Databases Workshop (2008) http://www-conf.slac.stanford.edu/xldb2008/

Cudré-Mauroux P, Kimura H, Lim K, Rogers J, Simakov R, Soroush E, Velikhov P, Wang DL, Balazinska M, Becla J, DeWitt DJ, Heath B, Maier D, Madden S, Patel JM, Stonebraker M, Zdonik SB (2009) A demonstration of SciDB: a science-oriented DBMS. PVLDB 2(2):1534–1537. http://www.vldb.org/pvldb/2/vldb09-76.pdf

Cudre-Mauroux P, Kimura H, Lim KT, Rogers J, Madden S, Stonebraker M, Zdonik SB (2010) Ss-db: A standard science DBMS benchmark. http://www-conf.slac.stanford.edu/xldb10/docs/ssdb_benchmark.pdf

Duggan J, Stonebraker M (2014) Incremental elasticity for array databases. In: International conference on management of data, SIGMOD 2014, Snowbird, pp 409–420. https://doi.org/10.1145/2588555.2588569

Ge T, Zdonik SB (2010) A*-tree: a structure for storage and modeling of uncertain multidimensional arrays. PVLDB 3(1):964–974. http://www.comp.nus.edu.sg/~vldb2010/proceedings/files/papers/R86.pdf

Leyshock P, Maier D, Tufte K (2013) Agrios: a hybrid approach to big array analytics. In: Proceedings of the 2013 IEEE international conference on big data, Santa Clara, pp 85–93. https://doi.org/10.1109/BigData.2013.6691558

Paradigm4 (2011) https://www.paradigm4.com/

Seering A, Cudré-Mauroux P, Madden S, Stonebraker M (2012) Efficient versioning for scientific array databases. In: IEEE 28th international conference on data engineering (ICDE 2012), Washington, DC (Arlington), pp 1013–1024. https://doi.org/10.1109/ICDE.2012.102

Soroush E, Balazinska M, Wang DL (2011) Arraystore: a storage manager for complex parallel array processing. In: Proceedings of the ACM SIGMOD international conference on management of data, SIGMOD 2011, Athens, pp 253–264. https://doi.org/10.1145/1989323.1989351

Stonebraker M (1986) The case for shared nothing. IEEE Database Eng Bull 9(1):4–9. http://sites.computer.org/debull/86MAR-CD.pdf

Stonebraker M, Madden S, Abadi DJ, Harizopoulos S, Hachem N, Helland P (2007) The end of an architectural era (it's time for a complete rewrite). In: Proceedings of the 33rd international conference on very large data bases, University of Vienna, pp 1150–1160. http://www.vldb.org/conf/2007/papers/industrial/p1150-stonebraker.pdf

Stonebraker M, Brown P, Poliakov A, Raman S (2011) The architecture of SciDB. In: Scientific and Statistical Database Management – proceedings of the 23rd international conference, SSDBM 2011, Portland, pp 1–16. https://doi.org/10.1007/978-3-642-22351-8_1

Stonebraker M, Brown P, Zhang D, Becla J (2013) SciDB: a database management system for applications with complex analytics. Comput Sci Eng 15(3):54–62. https://doi.org/10.1109/MCSE.2013.19

Wu E, Madden S, Stonebraker M (2013) Subzero: a fine-grained lineage system for scientific databases. In: 29th IEEE international conference on data engineering, ICDE 2013, Brisbane, pp 865–876. https://doi.org/10.1109/ICDE.2013.6544881

SDM Theory

▸ Spatial Data Mining

Search and Query Accelerators

Johns Paul[1], Bingsheng He[2], and Chiew Tong Lau[1]
[1]Nanyang Technological University, Singapore, Singapore
[2]National University of Singapore, Singapore, Singapore

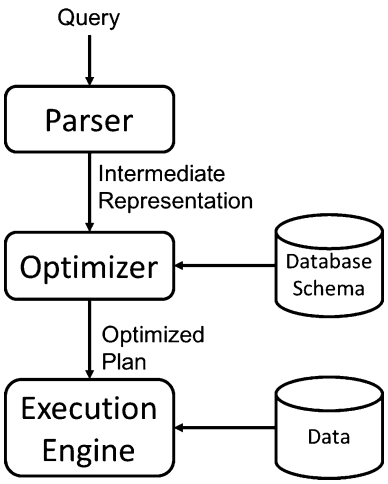
Search and Query Processing

Search and query processing involves translation of high-level queries or user requests into low-level operations which can be executed on physical hardware to retrieve relevant results from a database. The process generally involves conversion of the high-level query into an intermediate representation, optimization of the intermediate representation to generate an optimal evaluation plan and finally the execution of the optimized plan on physical hardware to retrieve relevant results (Markl 2009). Figure 1 shows the major steps involved in query processing for relational databases.

The performance of search and query processing is essential for our daily life and work productivity. Besides relational databases, everyday applications of search and query processing range from simple Google search to complex deep learning systems. Other common examples include online shopping, supporting internal operations of organizations, and even simple file searches on personal computers.

Accelerators for Query Processing

Most current query processing systems made use of CPUs to evaluate queries submitted by users. However, the explosive growth in the amount of available data and the huge demand for high-throughput and low-latency query processing have resulted in the adoption of specialized hardware devices to accelerate query processing. These accelerators act as an add-on to CPUs and



Search and Query Accelerators, Fig. 1 Query processing steps

are used only for running specialized tasks like query processing. Some of the most common accelerators used for query processing include GPUs, FPGAs, and x86 based accelerators like Xeon Phi.

GPU

Graphics processing units (GPUs) were initially designed to accelerate the rendering of images for video editing and gaming. However more recently, GPUs have evolved as a powerful accelerator for processing huge amounts of data in parallel. There are already a number of query processing systems available for GPUs including GPUQP (Fang et al. 2007), Ocelot (Heimel et al. 2013), GPL (Paul et al. 2016), MapD (MapD 2016), and Voodoo (Pirk et al. 2016).

Hardware Design

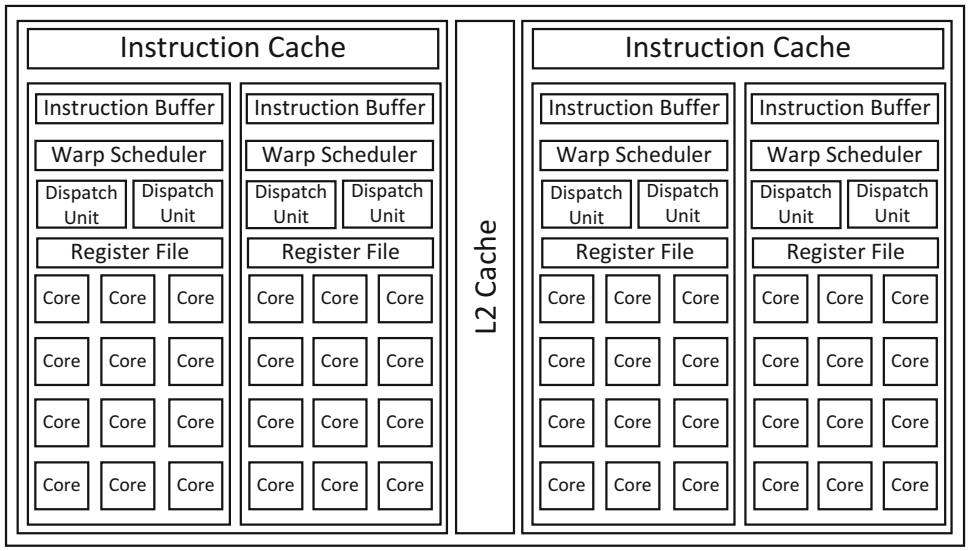
Similar to CPUs, GPUs also contain processing cores, registers, multiple levels of cache, and a global memory unit (the equivalent of the main memory for CPUs). However, GPUs differ from CPUs in terms of the design of each one of these components. Instead of a small number of complex cores available on the CPUs, GPUs use a large number of relatively simpler cores that work in a SIMD fashion to process large amounts of data. For example, the latest Pascal GPUs from NVIDIA contain more than 3500 individual cores which can work in parallel. This is more than 100× the number of cores in a modern CPU. GPUs are able to fit such a large number of cores in a single silicon die due to the relative simplicity of each individual core. These cores lack complex components such as branch prediction unit and even have much smaller cache than CPUs. However, they do have much larger number of registers and a higher bandwidth global memory unit, which is usually smaller (in size) than the main memory available on CPUs. The High Bandwidth Memory (HBM) units available in modern GPUs can provide close to 10× the bandwidth of the main memory accessible to CPUs. This high bandwidth and large number of cores help the modern Pascal GPUs from NVIDIA to achieve up to 9 TFLOPs of compute power using a single GPU. Figure 2 shows an overview of the internal architecture of the latest Pascal generation of GPUs from NVIDIA (2016).

Most modern GPUs are available as a separate hardware unit which can be connected to the CPU over the PCIe bus. Any data that needs to be processed by the GPU needs to be first copied to the GPU global memory over the PCIe bus. However, the 16 GB/s memory bandwidth of the current generation of PCIe bus (3.0) has proved to be a bottleneck to processing large amounts of data on GPUs. Hence, vendors like NVIDIA have started looking into better interconnects like NVLink which is capable of providing up to 5x the bandwidth of the PCIe bus.

Advantages and Limitations

The main advantage of GPUs for query processing tasks comes in the form of the availability of large number of parallel cores which work in a SIMD fashion. This is mostly because query processing operations largely involve applying the same operation to a large number of data items in parallel. Further, the availability of larger number of registers and the much higher global memory bandwidth of GPUs ensure fast data access to the large number of parallel cores, thus making it possible to achieve much better throughput than CPUs.

However, the need to use specialized languages like CUDA or OpenCL to program the



Search and Query Accelerators, Fig. 2 GPU internal architecture

GPUs has proved to be a major limiting factor in ensuring widespread adoption of GPUs. This is because existing query processing systems need to be rewritten in these languages to take advantage of GPUs. Further, the low bandwidth of the PCIe bus also dampens the overall performance of GPU-based systems due to the high overhead of data movement between the CPU main memory and GPU global memory. This coupled with the relatively small size of GPU memory limits the performance gains achieved by addition of GPUs to large query processing systems.

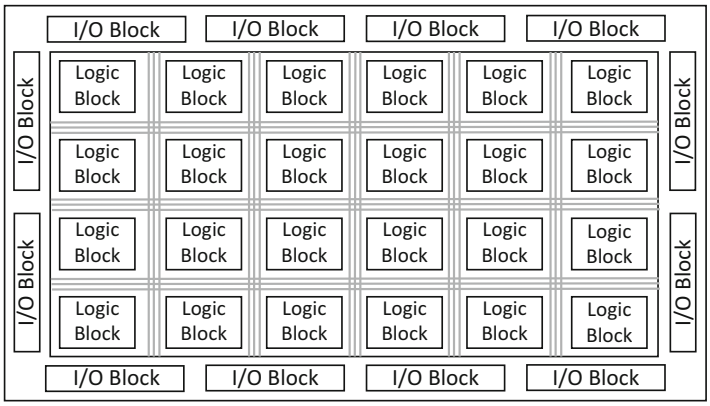
FPGA

Field-programmable gate arrays (FPGAs) are specialized semiconductor devices that consist of a collection of logic blocks which can be connected together using programmable interconnects based on the operation that needs to be executed on the FPGA. Due to its unique hardware design, FPGAs can achieve significantly better energy efficiency than other hardware accelerators and even CPUs when performing most tasks. This has led to services like Microsoft Bing search adopting FPGAs in their servers (Allison Linn 2016).

Hardware Design

An FPGA hardware consists of four main hardware units: logic blocks, registers, block RAM, and digital signal processors (DSPs). These hardware devices can be connected together based on the computation that needs to be executed on the FPGA. This means that FPGAs execute operations based on the interconnection in the hardware instead of decoding individual instructions as in the case of other hardware devices like CPUs or GPUs. Hence, FPGAs have the ability to process data more efficiently than other hardware devices making them significantly more energy efficient. FPGAs can be connected to CPUs over PCIe bus or Ethernet. Figure 3 shows an overview of architectural design of FPGAs.

In spite of being highly energy efficient, FPGA hardware usually falls behind other accelerators like GPUs when it comes to operating frequency and global memory bandwidth. Even though modern FPGAs have started adopting High Bandwidth Memory instead of traditional DDR RAM, these systems are still unable to achieve the same level of bandwidth as GPUs. FPGAs also lack additional on-chip cache which significantly impacts its performance when running programs written in high-level languages like OpenCL. Recent studies (Wang et al. 2016;



Search and Query Accelerators, Fig. 3
FPGA internal architecture

Woods et al. 2014) have demonstrated some promising results on accelerating query and search performance on FPGAs.

Advantages and Limitations

The major advantage of using FPGAs for query processing is its lower power consumption, which is very important in large data warehouses which spend a significant amount of money on power and cooling. Further, the use of hardware interconnections and the reliance on pipeline parallelism allow FPGAs to process data with much lower latency than other hardware devices like CPUs and GPUs. This is especially beneficial when querying data streams.

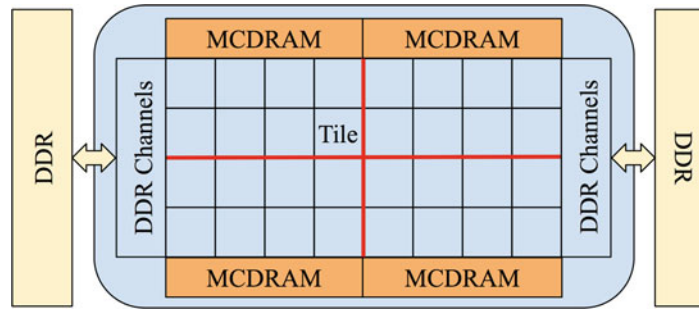
The major limitation of using FPGAs is the difficulty involved in programming them. Most FPGAs can be programmed only using low-level hardware description languages (HDLs) like Verilog or VHDL, making it extremely tedious to develop and maintain large-scale query processing systems on FPGAs. FPGA vendors have recently tried to address this issue by releasing OpenCL-based FPGAs which can be programmed using high-level languages like OpenCL. However, converting an operator written in OpenCL into an FPGA hardware implementation takes hours, and debugging these applications is extremely tedious. Further, the limited amount of resources available on the FPGA makes it difficult to fit a large number of operators on a single FPGA. All this makes it difficult to adopt FPGAs for systems that need to constantly adapt to user queries or changes in data streams.

Xeon Phi

The Xeon Phi accelerator was first introduced in 2012 by Intel as an alternative to accelerators like GPUs. The attraction of Xeon Phi is the use of traditional x86 cores which allow the use of standard programming languages and APIs such as OpenMP. This means that most existing query processing implementations designed for CPUs can be ported over to the Xeon Phi accelerator without the need for any modification. Due to the use of traditional x86 architecture, the latest generation of Xeon Phi can even run an operating system and work without CPUs.

Hardware Design

The Xeon Phi follows a many-core design that uses the x86 architecture. The accelerator tries to offer a device which contains much larger number of cores than modern CPUs while being architecturally similar to existing CPUs. The recent release of Xeon Phi, known as Knights Landing (Avinash Sodani 2016), contains over 70 cores which make use of SIMD intrinsics to process large amounts of data in parallel. These cores are arranged in a 2D mesh and have access to L1 and L2 cache, but it lacks an L3 cache. Further, due to power and area limitations, each individual core has much lower operating frequency and a much simpler pipeline design when compared to modern CPUs. The Xeon Phi accelerator also has access to high-speed memory stacks integrated into the same silicon die. Figure 4 shows the



Search and Query Accelerators, Fig. 4 Xeon Phi internal architecture

architectural design of the Knights Landing processor.

Further, in addition to operating as a standalone device (without the need for a CPU), the Xeon Phi can also be used in conjunction with traditional CPUs by connecting it to the CPU over the PCIe bus. Recent studies have demonstrated significant performance improvement by tuning and optimization on those platforms (Jha et al. 2015; Cheng et al. 2017).

Advantages and Limitations

The major advantage of the Xeon Phi accelerator is the much larger number of x86 cores available for processing data in parallel. This design enables the execution of existing query processing systems on the Xeon Phi accelerator without any modification and at the same time makes it possible to achieve much higher levels of parallelism than traditional CPUs.

However, the simpler design and lower operating frequency of each individual core limit the performance of the accelerator when executing single-threaded workload. This is especially problematic due to the relatively low performance of individual cores. The lack of an L3 cache can also have a significant negative impact in such cases. Further, connecting the Xeon Phi to the CPU over PCIe bus leads to the PCIe interconnect becoming a bottleneck for data transfer.

Future Hardware

A recent trend in query processing hardware seems to be the development of coupled architectures which consist both CPUs and specialized accelerators in the same physical die or on the same physical board. In such cases the accelerators are connected to CPUs using much faster interconnects like QPI or NVLink. Such a design allows the specialized hardware to access data residing in the system main memory without being bottlenecked by the lower-bandwidth PCIe bus. Some designs even allow the specialized hardware to access the L3 cache of CPUs, allowing for more fine-grained and faster movement of data between CPUs and hardware accelerators. Another recent trend in this area seems to be the emergence of database processing units (DPUs), which are specialized devices with hardware support for execution of relational operators (Wu et al. 2014).

References

Allison Linn (2016) The moonshot that succeeded: how bing and azure are using an AI supercomputer in the cloud. https://blogs.microsoft.com/ai/2016/10/17/the_moonshot_that_succeeded

Avinash Sodani (2016) Knights landing (KNL): 2nd generation Intel®Xeon Phi processor. https://www.alcf.anl.gov/files/HC27.25.710-Knights-Landing-Sodani-Intel.pdf

Cheng X, He B, Du X, Lau CT (2017) A study of main-memory hash joins on many-core processor: a case with intel knights landing architecture. In: Proceedings of the 2017 ACM on conference on information and knowledge management, CIKM '17. ACM, New York, pp 657–666. http://doi.acm.org/10.1145/3132847.3132916

Fang R, He B, Lu M, Yang K, Govindaraju NK, Luo Q, Sander PV (2007) GPUQP: query co-processing using graphics processors. In: Proceedings of the 2007 ACM SIGMOD international conference on management of data, SIGMOD '07. ACM, New York, pp 1061–1063. http://doi.acm.org/10.1145/1247480.1247606

Heimel M, Saecker M, Pirk H, Manegold S, Markl V (2013) Hardware-oblivious parallelism for in-memory

column-stores. Proc VLDB Endow 6(9):709–720. http://dx.doi.org/10.14778/2536360.2536370
Jha S, He B, Lu M, Cheng X, Huynh HP (2015) Improving main memory hash joins on intel xeon phi processors: an experimental approach. Proc VLDB Endow 8(6):642–653. http://dx.doi.org/10.14778/2735703.2735704
MapD (2016) The world's fastest platform for data exploration. http://go.mapd.com/rs/116-GLR-105/images/MapD%20Technical%20Whitepaper%20Summer%202016.pdf
Markl V (2009) Query processing (in relational databases). Springer US, Boston, pp 2288–2293. https://doi.org/10.1007/978-0-387-39940-9_296
NVIDIA (2016) NVIDIA Tesla P100. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf
Paul J, He J, He B (2016) GPL: a GPU-based pipelined query processing engine. In: Proceedings of the 2016 international conference on management of data, SIGMOD '16. ACM, New York, pp 1935–1950. http://doi.acm.org/10.1145/2882903.2915224
Pirk H, Moll O, Zaharia M, Madden S (2016) Voodoo – a vector algebra for portable database performance on modern hardware. Proc VLDB Endow 9(14):1707–1718. http://dx.doi.org/10.14778/3007328.3007336
Wang Z, Paul J, Cheah HY, He B, Zhang W (2016) Relational query processing on opencl-based fpgas. In: 2016 26th international conference on field programmable logic and applications (FPL), pp 1–10. http://dx.doi.org/10.1109/FPL.2016.7577329
Woods L, István Z, Alonso G (2014) Ibex: an intelligent storage engine with support for advanced sql offloading. Proc VLDB Endow 7(11):963–974. http://dx.doi.org/10.14778/2732967.2732972
Wu L, Lottarini A, Paine TK, Kim MA, Ross KA (2014) Q100: the architecture and design of a database processing unit. SIGARCH Comput Archit News 42(1):255–268. http://doi.acm.org/10.1145/2654822.2541961

Secure Big Data Computing in Cloud: An Overview

Sambit Kumar Mishra, Sampa Sahoo, and Bibhudatta Sahoo
National Institute of Technology Rourkela, Rourkela, India

Definitions

Cloud computing: It is a system model that provides services on demand from a shared pool of resources (CPU, main memory, secondary storage, bandwidth, etc.) without violating service level agreement (SLA) and maintaining a certain level of quality of service (QoS). *SLA*: The agreement done between the cloud service provider and cloud user before accessing the services.

Big data: It is an emerging area applied to store, manage, and analyze the data whose volume is large.

Virtualization: It is a technology that facilitates multiple virtual machines (VMs) over a single physical machine (host) with the help of a hypervisor or virtual machine monitor (VMM).

Virtual machine: A VM is a software program that emulates a real (physical) computing environment where an operating system is installed and can run different applications.

Introduction

The big data (explosion of data) problem aroused over the last 5–8 years as a result of the wide area network access, proliferation of next-generation applications, and advent of social media. Diverse sectors like finance, retail, smart sensor networks (IoT), physical and life science, etc., generate large data sets that need significant storage and computational implications. The bursty nature of data set spurred the use of cloud computing, where users can rent infrastructure on a "pay-as-you-go" basis. Virtualization technology, the basis of cloud computing, customized hardware and computational power of physical machines (PM), thus creating an illusion of several machines (virtual machines (VM)). Virtualization ensures the maximum of resource utilization and capital investment (Mishra et al. 2018b). A VM is a software application that emulates a physical computing environment, and multiple VMs can run on a single physical machine. Thus, virtualization reduces large capital infrastructure and maintenance cost. Cloud computing provides a scalable and cost-efficient solution to the big data challenge. Traditional computing (grid) technology had put a significant effort on resilience and robustness instead of solving the actual problem.

Modern big data technologies use scalable and cost-efficient methods to overcome such limitations.

The big data technology, though extremely powerful, was built for the technically savvy and needs applications to be parallelized in nature (ODriscoll et al. 2013). Big data with cloud technology makes several applications (e.g., biological problem) into reality to provide solutions. The implication of cloud computing to big data is due to the scalable and fault tolerance features of cloud computing toward big data computing. Addressing big data is a challenging and time-demanding task that requires a sizeable computational infrastructure to ensure reliable data processing and analysis. Since users are transmitting their data through the Internet, it gives rise to privacy issues, concerning profiling, stealing, and loss of control (Puthal et al. 2015). Although cloud computing transformed modern ICT technology, there exist security threats that can be elaborated by the volume, velocity, and variety of big data (Hashem et al. 2015).

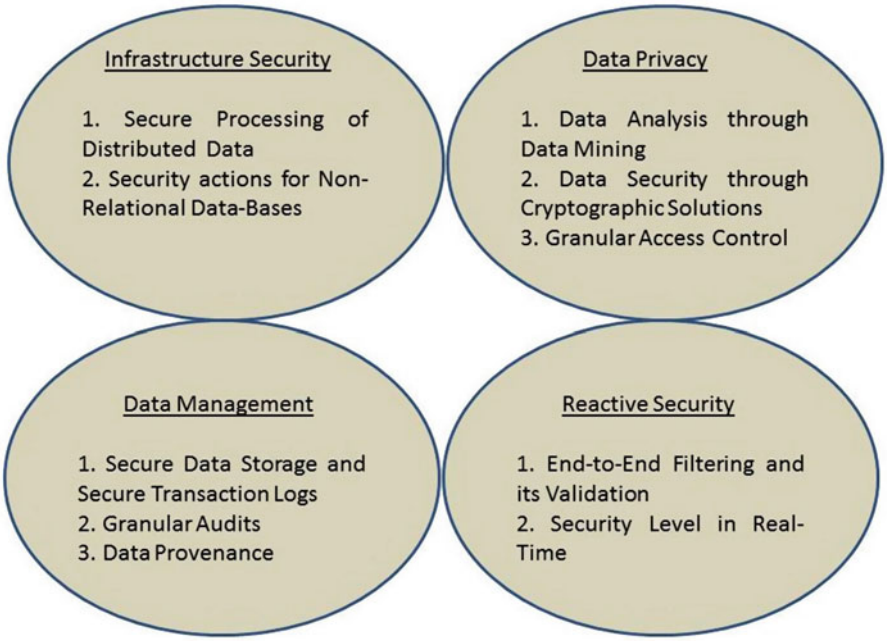
Some drawbacks associated with utilization of cloud computing are (1) data transfer rate: data from some applications may take long transmission time, even a week (e.g., genomic data) over the Internet. (2) Security and protection: Even though cloud computing is capable of handling big data sets, there is a loophole in terms of security and protection of data. Many applications such as cloud-based health-care system must provide enough security and protection of sensitive data from intruders. (3) Data tenancy: cloud infrastructures provide very little capability on data, application, and service back to an on-premise IT environment.

The big data processing capabilities of cloud computing generates a new area of computing system research. The requirement of secured big data computing in cloud is more essential due to the rise of the Internet of things (IoT). Gartner estimates that 26 billion of IoT devices (connected in mobiles, cars, TVs, security systems, etc.) will be installed by 2020 (Rivera and van der Meulen 2014). This will be responsible for a significant security challenge; however, it presents huge benefits for end users.

The big data not only deal with terabytes of data but also need to handle the issue of managing data under a traditional framework. The high-speed connectivity era permit movement of large sets of data that may contain sensitive information like credit/debit card numbers, addresses, and other details, raising data security concerns. Security issues in the cloud are a major concern for businesses and cloud providers today. The remaining of the paper is arranged as follows. The next section outlines an overview of secured challenges of big data computing in cloud; after that the next section reviews various solutions for security challenges in cloud followed by the conclusion and future direction.

Secured Challenges of Big Data Computing in Cloud

The complex infrastructure of cloud computing system increases various challenges for the IT industries and researchers. The volume of data collection, storage, and processing in the cloud system increases day by day, which brings new challenges in terms of the information security. Due to the stretching of security mechanisms out of the perimeter of the requirements, general firewalls cannot be used in the big data infrastructure (Kune et al. 2016). The challenges for secure big data computing may be organized into different big data aspects such as infrastructure security, data privacy, data management, and reactive security (Moura and Serrão 2016). Each of these mentioned security aspects face different security challenges as presented in Fig. 1. The security functions are compelled to operate over the heterogeneous composition using big data. Different technologies introduced abstraction that can enable the big data secure services on top of the heterogeneous infrastructure and separate the control from the system infrastructure. The production rate of data has been rising exponentially due to various sources like sensors. Thus, data privacy plays an important role due to the risk of erroneous data in every field. For example, the patient data; the content of patient medical record possess high-risk on the health-care system. The



Secure Big Data Computing in Cloud: An Overview, Fig. 1 Four different aspects of secure challenges of big data computing in cloud

big data computing has a set of risk areas that includes the ownership and classification of data, the process of creation, and collection of data. It would be more challenging in terms of research point of view to develop techniques to delegate encrypted data so that third parties can analyze it.

Some sensitive data require more security for the sake of the client; there should be an agreement (i.e., specified in SLA) upon the location of data, as its data may be considered illegal for others and lead to prosecution. Data encryption is required to solve security and privacy issues. Therefore, researchers have focused on the generation of new systems that must ensure the quick access of data where there is no effect of encryption on processing. To improve the system performance, various mechanisms (task allocation, VM consolidation, etc.) has to be improvised (Mishra et al. 2018a). To optimize the performance parameters like energy consumption, makespan, throughput, SLA violation, etc., the profit for the CSP as well as for the cloud user reduced (Mishra et al. 2018b). Due to the complexity of the system with the aim of optimizing various performance parameters, the security level also has to be improved, which leads to a huge cost.

Solutions to the Security Challenge

For an efficient big data management and processing, the cloud system needs a high-speed transport mechanism that addresses two significant bottlenecks: (1) the degradation in the speed of WAN transfer that occurs over distance through traditional transfer protocols and (2) the last foot bottleneck caused by the HTTP interfaces in the cloud data center to the underlying object based cloud storage.

The new era of computing technology allows companies to store and analyze the vast amount of data (e.g., company, business, customer). So, it becomes a challenging task to make the data secure. To make the big data secure, various techniques are used for encryption, honeypot detection, logging, etc. The challenge of detecting and preventing advanced threats and malicious intruders must be solved using big data style analysis. With the rise in the use of big data,

companies must have to deal with both security and privacy issues. The big data issues are most acutely felt in certain industries, such as telecoms, web marketing and advertising, retail and financial services, and certain government activities. So, resource allocation and memory management algorithms used in the cloud for big data also have to be secure. Numerous data mining techniques can be used in the malware detection in cloud (Inukollu et al. 2014).

Kune et al. have elaborated the differences between the traditional data warehousing and big data and also discussed various solutions for the security issues in big data computing (Kune et al. 2016). Several areas of big data computing (i.e., scientific explorations, health care, governance) have been explained in Kune et al. (2016). The collected data from different sensors are extracted by analyzing those data for societal benefits.

Attackers also keep inventing new ideas of attacking to a secure system. Other issues like ransomware profoundly affect a company's reputation and resources, denial of service attacks, phishing attacks, and cloud abuse. Globally, 40% of businesses experienced a ransomware incident during the past year. Both clients and cloud providers have their own share of risks involved when agreeing on cloud solutions. Insecure interfaces and weak APIs can give away valuable information to hackers, which can be misused. Fraud detection patterns, encryptions, and smart solutions are immensely valuable to combat attackers. At the same time, it is the responsibility of the user to own data and keep it safe while looking for intelligent answers that can assure a steady ROI (return on investment) as well (Riaz 2017).

Conclusion and Future Directions

This chapter presents some of the most significant security and privacy issues that affect big data computing in cloud environment and also discusses the information security, methodologies, and tools to provide security and privacy to the system. It also presents some relevant solutions for these security issues. This chapter does not point to directions and technologies that contribute to solve some of the relevant security issues.

References

Hashem IAT, Yaqoob I, Anuar NB, Mokhtar S, Gani A, Khan SU (2015) The rise of big data on cloud computing: review and open research issues. Inf Syst 47:98–115

Inukollu VN, Arsi S, Ravuri SR (2014) Security issues associated with big data in cloud computing. Int J Netw Secur Appl 6(3):45

Kune R, Konugurthi PK, Agarwal A, Chillarige RR, Buyya R (2016) The anatomy of big data computing. Softw: Pract Experience 46(1):79–105

Mishra SK, Putha D, Rodrigues JJ, Sahoo B, Dutkiewicz E (2018a) Sustainable service allocation using meta-heuristic technique in fog server for industrial applications. IEEE Trans Ind Inf

Mishra SK, Puthal D, Sahoo B, Jena SK, Obaidat MS (2018b) An adaptive task allocation technique for green cloud computing. J Supercomput 74:370–385

Moura J, Serrão C (2016) Security and privacy issues of big data. arXiv preprint arXiv:160106206

ODriscoll A, Daugelaite J, Sleator RD (2013) Big data, Hadoop and cloud computing in genomics. J Biomed Inform 46(5):774–781

Puthal D, Sahoo B, Mishra S, Swain S (2015) Cloud computing features, issues, and challenges: a big picture. In: Computational intelligence and networks (CINE), 2015 international conference on. IEEE, pp 116–123

Riaz K (2017) http://bigdata-madesimple.com/big-data-and-cloud-computing-challenges-and-opportunities/

Rivera J, van der Meulen R (2014) Gartner says the internet of things will transform the data center. Retrieved 5 Aug 2014

Security and Privacy Aspects of Semantic Data

Sabrina Kirrane
Vienna University of Economics and Business, Vienna, Austria

Synonyms

Security and privacy for the resource description framework

Definitions

Access control is a mechanism used to restrict access to data or systems, based on rules that grant *subjects* (e.g., individuals, groups, roles) *access rights* to *resources* (e.g., data or systems) (Sandhu and Samarati 1994). Enforcement is usually broken into two stages: *authentication* and *authorization*. Authentication involves the verification the data subjects identity or attributes, whereas authorization is a mechanism used to determine if the requester (i.e., the subject) has the access rights necessary to carry out the request.

Encryption is an effective means of ensuring the confidentiality and integrity of information stored locally or transferred over a network (Menezes et al. 1996). Encryption involves the translation of data into an unintelligible form through the use of a secret key. Decryption is the process of restoring data to its original form through the use of a key (which may or may not be the same as the key used to encrypt the data). Encrypted data is referred to as cipher or cipher text, whereas unencrypted data is commonly known as plain text.

Trust mechanisms are used to verify the validity of a claim (e.g., the identity/attributes of an individual or the correctness of data). The most widely used trust mechanisms include *policies* and *reputation* (Artz and Gil 2007). Policies are used to govern the exchange of credentials that are often certified by trusted third parties. Reputation mechanisms take the form of provenance information and metrics that are calculated from previous actions and behaviors. Where no such data is available, trust may be established via referral from other trusted parties.

Anonymization involves the removal of personally identifiable information from datasets. One of the most well-known anonymization techniques, k-anonymity, involves the use of suppression (i.e., masking sensitive data) and generalization (i.e., choosing broader classification terms for sensitive data), in order to group individuals into equivalence classes, whereby each individual in a class is indistinguishable from *k-1* other individuals (Samarati and Sweeney 1998).

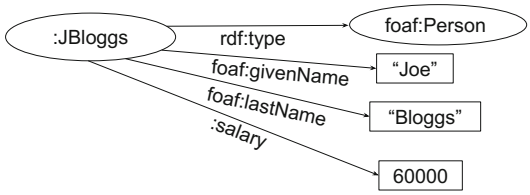
Overview

The Resource Description Framework (Manola and Miller 2004) is designed to facilitate data integration and reuse by representing distributed data in a machine-readable format. RDF *vocabularies* (otherwise known as *ontologies*) are collections of RDF triples that can be used to describe both schema and instance data. Each triple, which is composed of a *subject-predicate-object* expression, asserts a binary relationship between two pieces of information. *Internationalized Resource Identifiers (IRIs)* and *literals* are used to represent information, which can be either physical or abstract in nature. The *RDFSchema* (RDFS) ontology (Brickley and Guha 2014) is composed of a set of classes and properties commonly used to describe RDF data. RDFS does not describe the structure of an RDF graph but rather provides a framework that can be used to denote classes, properties, and relations. Vocabularies are often placed in a common *namespaces*, referenced via *prefixes*. In the examples that follow, the default prefix `:` is used to denote an example enterprise ontology <http://example.org/ex/>. In addition, well-known `rdf` and `foaf` prefixes are used for the RDF built-in vocabulary and FOAF social network ontology, respectively. Example 1 demonstrates how RDF can be used to represent information pertaining to `Joe Bloggs`.

Example 1 (RDF triples) The following triples state that the entity `:JBloggs` is a person whose first name is `Joe`, last name is `Bloggs`, and salary is `60,000`:

```
:JBloggs rdf:type foaf:Person.
:JBloggs foaf:givenName "Joe".
:JBloggs foaf:lastName "Bloggs".
:JBloggs :salary 60000.
```

An *RDF graph* is a finite set of RDF triples, with subjects and objects represented as nodes and predicates represented as edges. Figure 1 demonstrates how the triples in Example 1 converge to form a graph, with IRIs represented as ovals and literals represented as rectangles. A collection of RDF graphs, which can include a



Security and Privacy Aspects of Semantic Data, Fig. 1 Triples represented as an RDF graph

default graph and one or more named graphs, is known as an RDF dataset.

In a recent survey by Fernandez Garcia et al. (2016), the authors analyzed topics appearing in papers that were published in Semantic Web conference proceedings and journals from 2006 to 2015 inclusive. The results of the conducted text analysis confirmed that traditional Semantic Web topics, such as knowledge representation, data management, system engineering, searching, browsing and exploration, and data integration, dominated the field up to 2015.

According to Fernandez Garcia et al. (2016), although topics relating to security and privacy have shown a minor increase over the years, the topics remain under-represent in comparison to traditional topics.

Much of the early research on security and privacy in the context of the Semantic Web focused on using RDF to represent existing access control models and standards and demonstrating how the technology could be used to develop general policy languages. Later the focus moved to the development of access control strategies for RDF and the Semantic Web. Other popular topics over the years include demonstrating how existing encryption mechanisms can be used to protect RDF data and establishing trust mechanisms for the Semantic Web. More recently, the landscape has broadened to include the encryption and anonymization of RDF data.

Key Research Findings

The goal of this section is to introduce the reader to key research findings in relation to Semantic Web security and privacy, and as such it focuses on the predominant topics within the community, namely, access control, encryption, trust, and anonymization.

Access Control

Access Control (AC) for the RDF data model has predominately focused on using patterns to specify authorizations, enabling inference based on semantic relations between policy entities, and demonstrating how RDF can be used to form general policy languages.

Reddivari et al. (2005) demonstrate how access control rules can be used to manage access to an RDF store. Two predicates `permit` and `prohibit` are used to grant and deny access rights based on common database actions (e.g., `INSERT, DELETE, SELECT`) to one or more triples using triple patterns (cf. Example 2). A triple pattern is composed of an RDF triple with optionally a variable (denoted by a ?) in the subject, predicate, and/or object position.

Example 2 (AC rules with triple patterns) The following rule states that a subject `Alice` can create instances of any class (denotes as `?y`) if there is an assertion that subject `Alice` created that class.

```
permit(INSERT(Alice,
(?x,rdf:type,?y)))
:- createdNode(Alice,?y)
```

Jain and Farkas (2006) build on the approach proposed by Reddivari et al. (2005), by demonstrating how RDFS entailment rules can be used to derive authorizations for inferred triples. While Kirrane et al. (2013) demonstrate how authorizations based on quad patterns (where the fourth element denotes the named graph) can be used to enforce Discretionary Access Control (DAC), where users can pass their access rights on to other users. Like Jain and Farkas (2006), the authors derive access rights for derived data using RDFS entailment rules.

When it comes to access control enforcement, typical enforcement strategies involve filtering unauthorized data based on access control policies and executing queries against the

filtered dataset or using query rewriting techniques to inject access control filters into queries.

Dietzold and Auer (2006) and Gabillon and Letouzey (2010) demonstrate how graph patterns (i.e., sets of triple patterns) constrained by a `WHERE` clause can be used to create a new dataset that only contains data the subject is permitted to access. The authorized dataset is created using SPARQL the standard query language for RDF (Seaborne and Prud'hommeaux 2008). Essentially, authorizations contain filters that refer to sparql `CONSTRUCT` queries that are used to generate the authorized dataset. In Gabillon and Letouzey (2010) a rule such as `Permit(Alice, SELECT, foafview.txt)` can be used to permit subject `Alice`, access right `SELECT` on resource `foafview.txt`. The resource `foafview.txt` simply contains a `CONSTRUCT` query such as that presented in Example 3). When a requester submits a query, a new dataset is created based on the matched authorizations. The query is executed against the new dataset, which only contains data that the requester is permitted to access.

Example 3 (Construct view) The following query creates a dataset that contains all data relating to people with `Bloggs` as a last name.

```
CONSTRUCT {?x ?p ?y}
WHERE {
?x ?p ?y .
?x foaf:lastName "Bloggs"}
```

An alternative enforcement strategy proposed by Abel et al. (2007) uses query rewriting to create bindings for variables that are subsequently added to the query `WHERE` clause. In the case of negative authorizations, the bindings are added to a MINUS clause, which is appended to the query. A simple SPARQL `SELECT` query is presented in Example 4, and sample rewritten queries containing positive and negative filters are presented in Examples 5 and 6, respectively. When a requester submits a query, the enforcement framework rewrites the query according to the matching authorizations, and the rewritten query is subsequently executed against the new dataset, ensuring that the requester is only returned data that they are permitted to access.

Example 4 (SELECT query) The following query returns all data.

```
SELECT *
WHERE { ?s ?p ?o }
```

Example 5 (Positive filter) The following query, which contains a positive filter, only returns the information for `:JBloggs`.

```
SELECT *
WHERE { ?s ?p ?o .
FILTER ( ?s = :JBloggs ) }
```

Example 6 (Negative filter) The following query, which contains a negative filter, returns everything except the `:salary` information.

```
SELECT *
WHERE { ?s ?p ?o .
MINUS { ?s ?p ?o .
FILTER ( ?p = :salary ) }}
```

In addition to the access control mechanisms described above, there have been a number of standardization initiatives that could be used to limit access to RDF data. *Web Identity and Discovery (WebID)* (Sporny et al. 2011) is a mechanism that can be used to uniquely identify and authenticate a person, company, organization, or other entity, by means of a Uniform Resource Identifier (URI), while *Web Access Control (WAC)* W3C (n.d.) is an RDF vocabulary and access control framework that can be used for policy specification and enforcement. Both Villata et al. (2011) and Sacco and Passant (2011) extend WAC to cater for access control over the RDF data model. Using the extended vocabularies, it is possible to associate access control with individual RDF resources (subjects, predicates, and objects) and also collections of RDF resources (named graphs). In addition, the authors extend the vocabulary to cater for a broader set of access privileges.

An alternative policy language, called the Open Digital Rights Language (ODRL) (Iannella

and Villata 2018), is a general rights language that can be used to define rights for limiting access to digital resources. When it comes to ODRL and RDF, primary research efforts to date focus on demonstrating how ODRL can be used to express a variety of access policies (Steyskal and Polleres 2014; Steyskal and Kirrane 2015) and using ODRL vocabularies to specify RDF licenses (Cabrio et al. 2014).

A comprehensive survey of existing access control strategies for RDF is presented in Kirrane et al. (2017).

Encryption

Encryption techniques for RDF have received very little attention to date, with work primarily focusing on the partial encryption of RDF data, the querying of encrypted data, and the signing of RDF graphs.

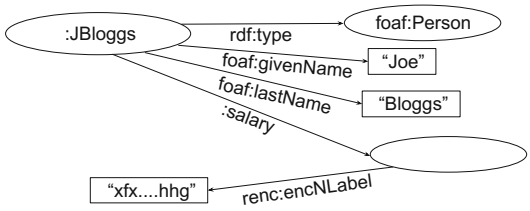
Giereth (2005) demonstrates how public-key encryption can be used to partially encrypt RDF fragments (i.e., subjects, objects, or predicates). The ciphertext and the corresponding metadata (algorithm, key, hash, etc.) are represented using a literal that they refer to as an encryption container. When only the object is encrypted, the object part of the triple is replaced with a blanknode (i.e., an anonymous resource), and a new statement is created with the blanknode as the subject, the encryption container as the object, and a new `renc:encNLabel` as the predicate (cf. Fig. 2). The treatment of encrypted subjects is analogous. The encryption of predicates is a little more difficult, as reification (a technique used to make statements about resources) is needed to associate the new blanknode with the relevant subject, object, and encryption container.

Rather than simply storing the encrypted data and metadata in a literal, Gerbracht (2008) discusses now the metadata that can be represented using multiple triples using their crypto ontology. The encrypted element or subgraph is replaced with a new unique identifier, and new statements are added for the encrypted data and the corresponding metadata (cf. Fig. 3).

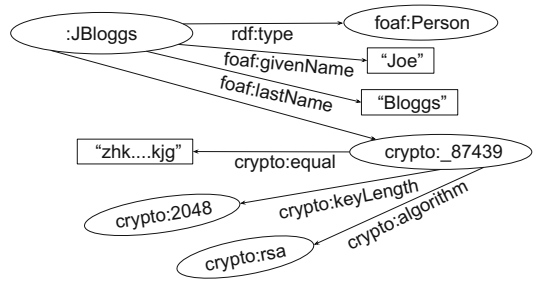
Kasten et al. (2013) in turn focus on querying encrypted data. In the proposed framework, each triple is encrypted eight times according to the eight different triple pattern binding possibilities. The proposed approach allows for graph pattern queries to be executed over the ciphertext, at the cost of storing multiple ciphers for each statement. An alternative approach by Fernández et al. (2017) demonstrates how functional encryption can be used to generate query keys based on triple patterns, whereby each key can decrypt all triples that match the corresponding triple pattern. Other work by Kasten et al. (2014) investigates enabling of the signing of graph data at different levels of granularity.

Trust

In 2007, Artz and Gil (2007) conducted a survey of existing trust mechanisms in computer science in general and the Semantic Web in particular. The authors highlight that traditional approaches focused primarily on authentication via assertions by third parties; however, in later years, the topic evolved to include historical interaction data, the transfer of trust from trusted entities,



Security and Privacy Aspects of Semantic Data, Fig. 2 Partially encrypted RDF graph



Security and Privacy Aspects of Semantic Data, Fig. 3 Partially encrypted RDF graph and metadata

and decentralized trust mechanisms (e.g., voting mechanisms or other consensus decision-making mechanisms).

Existing work on trust and semantic data focuses primarily on demonstrating how existing trust metrics can be applied to Semantic Web Data, the development of policy languages to support trust and negotiation, and the identification of trust architectures and frameworks.

Ding et al. (2003, 2005) discuss how various trust mechanisms can be combined in order to determine the reliability of information published on the Semantic Web. The proposed trust mechanism combines historical data, information obtained directly from trusted semantic agents, and information based on referrals from trusted agents.

The PeerTrust policy language and framework (Gavriloaie et al. 2004) demonstrate how together semantic annotations and access control rules can be used to support automated trust negotiation and access control. An alternative policy language called Protune is described in Bonatti and Olmedilla (2005, 2007). Although Protune is in fact a general policy language, the authors focus primarily on trust negotiation and policy explanations.

Bizer and Oldakowski (2004) propose a trust architecture that combines reputation, content, and context-based trust mechanisms. The *information integration layer* aggregates data from several sources and adds the relevant provenance metadata. The *repository layer* is used to store the information and associated metadata in named graphs. The *query and trust evaluation layer* uses trust policies to make trust decisions. Here the authors rely on a query language TriQL.P that is used to return the query results together with a justification tree that can be used to understand how the query results fulfill the trust requirement. Finally the *application and explanation layer* receives requests and provides the trust decision together with the relevant explanations.

More recently, Laufer and Schwabe (2017) propose a framework that can be used to describe the trust process. Inputs to be considered include the data and associated metadata and contextual information relating to the action that needs to be taken, together with trust policies specified by the agent. Like Ding et al. (2003), the trust process relies on historical data, along with direct and indirect sources of information.

Anonymization

The anonymization of RDF data has recently emerged as a popular research topic, with work to date focusing on the application of k-anonymity (Samarati and Sweeney 1998) or differential privacy (Dwork 2006) to RDF data.

Radulovic et al. (2015) propose a framework called k-RDFanonymity, which includes an anonymization model, generalization and suppression operations, and distortion metrics that are specifically tailored for the anonymization of RDF data. The authors highlight the fact that RDF differs from tabular data as identifiers, quasi-identifiers, and sensitive attributes can appear in the subject, predicate, and object positions. Additionally the anonymization needs to be able to handle data represented as literals and IRIs. In the proposed model, generalization involves the replacement of resources (i.e., literals or IRIs) with more general resources based on domain hierarchies, while suppression involves either the removal or replacement of resources.

Heitmann et al. (2017) build on this work to ensure protection against neighborhood attacks. The proposed approach, which is known as k-RDF-Neighborhood anonymity, ensures that one-hop neighbors of an anonymized resource are indistinguishable from k-1 one-hop neighbors of other resources.

Other work in relation to RDF anonymization includes adopting graph or statistical database approaches. Lin (2016) takes inspiration from existing graph isomorphism-based anonymization techniques, discussing their suitability for RDF data from both a security and a computational complexity perspective. Whereas, Silva et al. (2017) explore the application of existing differential privacy mechanism to RDF data and propose a framework that can be used to compute differential privacy parameters.

Examples of Application

Semantic Web technologies have a solid foundation in open standards as evidenced by the various World Wide Web Consortium (W3C) recommendations; however, the layers of the Semantic Web technology stack (Berners-Lee 2000) that relate to security and privacy (i.e., unifying logic, proof, trust, and cryptography) are still very immature. Although the application of the key research findings described in the previous section is still very exploratory, several of the articles are guided by real-world use cases and practical applications.

For instance, the Protune policy language (Bonatti and Olmedilla 2005, 2007), which was developed by the Research Network of Excellence on Reasoning on the Web, known as REWERSE, was tasked with building the foundations for the advanced of Web systems and applications by developing inter-operable reasoning languages.

Fernández et al. (2017) are motivated by a real word use case that involves the combination of open and closed data in a data market scenario. In order to demonstrate the suitability of the proposed encryption mechanism, the authors conduct a performance evaluation over two real-world datasets: Jamendo a large dataset containing licensed music and the AEMET metereological dataset.

Although the initial evaluation of the trust framework proposed by Ding et al. (2003) was conducted using simulated data, the authors later discussed how trust mechanisms could be used in the context of homeland security, in order to identify suspicious individuals, relationships, activities, or events (Ding et al. 2005). Similarly, Laufer and Schwabe (2017) describe how the proposed trust framework can be used to evaluate the trustworthiness of claims in relation to political agents in Brazil coming from a variety of public sources (e.g., news stories, tweets, social media postings).

Existing work on anonymization appears to be less applied that the other topics with authors simply motivating their work by referring to privacy concerns in domains, such as healthcare and energy (cf. Radulovic et al. 2015).

Future Directions for Research

From a community perspective, it is well known that privacy is a multidisciplinary research area, which brings with it the need for closer collaboration between computer scientists, humanities, and social scientists and legal scholars. Although initiatives such as the *Society, Privacy and the Semantic Web – Policy and Technology* (PrivOn) workshop, which was collocated with the International Semantic Web Conference (ISWC) from 2013 to 2017, provides a forum for multidisciplinary research, stronger collaboration between different research communities is still needed.

From a technical perspective, there is a need for more applied work and a focus on attacker models across all privacy and security topics. Additionally there are many open research questions concerning the topics presented in this paper, several of which are outlined below.

When it comes to information security, there is still no standard access control strategy for the Semantic Web. Considering the array of access control specification and enforcement mechanisms proposed to date, a necessary first step is to develop a framework that can be used to evaluate existing offerings in terms of correctness, completeness, and robustness.

As for encrypted RDF, it is still not possible to execute complex queries and computations over encryption RDF data. One interesting avenue for future work is the application of homomorphic encryption to RDF; however, it brings with it performance and scalability issues that still need to be tackled. Another open research topic is the simplification of key management for multiple datasets, federated querying over encrypted data, and providing support for the revocation of existing keys.

In terms of trust, a recent article by Beek et al. (2016) highlights several issues with respect to the quality of existing data and datasources, claiming that the Semantic Web is neither traversable nor machine-processable and consequently arguing that the Semantic Web needs centralization. A counterargument, which is more in keeping with the goals of the Semantic

Web, would be to argue for the application of trust mechanisms into the fabric of the Semantic Web, which could be brought about by the realization of the upper layers and vertical layers of the Semantic Web technology stack.

Anonymization is a relatively new topic within the Semantic Web community with works primarily focusing on k-anonymity. However, it is well known that k-anonymity is prone to homogeneity and background knowledge attacks. Common extension mechanisms include l-diversity (Li et al. 2007), which ensures that sensitive attributes within an equivalence class are suitably different, and t-closeness, which ensures that the distribution of each equivalence class is representative of the distribution of the entire dataset (Machanavajjhala et al. 2006).

Other promising privacy and security research directions that remain underdeveloped and as such have not been presented in this article include usage control, which is defined as an extension of access control that enables data publishers to dictate not only who can access their data but also what they are permitted to do with this data (Bonatti et al. 2017). Related topics include transparency, which involves being open with respect to data processing and sharing, and accountability, which involves making data consumers responsible for their actions. Here, interesting avenues for future work include the adoption and extension of non-repudiation and fair exchange protocols.

Cross-References

- Big Data for Cybersecurity
- Privacy-Aware Identity Management
- Privacy-Preserving Data Analytics
- Privacy-Preserving Record Linkage

References

Abel F, De Coi J, Henze N, Koesling A, Krause D, Olmedilla D (2007) Enabling advanced and context-dependent access control in RDF stores. In: The semantic web. Lecture notes in computer science, vol 4825. Springer, Berlin/Heidelberg, pp 1–14. https://doi.org/10.1007/978-3-540-76298-0_1

Artz D, Gil Y (2007) A survey of trust in computer science and the semantic web. Web Semant Sci Serv Agents World Wide Web 5(2):58–71

Beek W, Rietveld L, Schlobach S, van Harmelen F (2016) Lod laundromat: why the semantic web needs centralization (even if we don't like it). IEEE Internet Comput 20(2):78–81

Berners-Lee T (2000) Semantic web – xml2000. https://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html. Accessed 13 Jan 2018

Bizer C, Oldakowski R (2004) Using context-and content-based trust policies on the semantic web. In: Proceedings of the 13th international World Wide Web conference on alternate track papers & posters. ACM, pp 228–229

Bonatti P, Olmedilla D (2005) Driving and monitoring provisional trust negotiation with metapolicies. In: Sixth IEEE international workshop on policies for distributed systems and networks, pp 14–23

Bonatti PA, Olmedilla D (2007) Rule-based policy representation and reasoning for the semantic web. In: Proceedings of the third international summer school conference on reasoning web, RW'07. Springer, pp 240–268. http://dl.acm.org/citation.cfm?id=2391482.2391488

Bonatti P, Kirrane S, Polleres A, Wenning R (2017) Transparent personal data processing: the road ahead. In: International conference on computer safety, reliability, and security. Springer, London/New York, pp 337–349

Brickley D, Guha R (2014) RDF schema 1.1. W3C recommendation, W3C. Available at http://www.w3.org/TR/2014/REC-rdf-schema-20140225/Overview.html

Cabrio E, Aprosio AP, Villata S (2014) These are your rights a natural language processing approach to automated RDF licenses generation. In: The semantic web: trends and challenges. Springer, Cham, pp 255–269

Dietzold S, Auer S (2006) Access control on RDF triple stores from a semantic wiki perspective. In: Proceedings of the ESWC'06 workshop on scripting for the semantic web

Ding L, Zhou L, Finin TW (2003) Trust based knowledge outsourcing for semantic web agents. In: Proceedings IEEE/WIC international conference on web intelligence, pp 379–387

Ding L, Kolari P, Finin T, Joshi A, Peng Y, Yesha Y (2005) On homeland security and the semantic web: a provenance and trust aware inference framework. In: AAAI spring symposium: AI technologies for homeland security, pp 157–160

Dwork C (2006) Differential privacy. In: Proceedings of the 33rd international conference on automata, languages and programming – volume part II, ICALP'06. Springer, Berlin/Heidelberg, pp 1–12. https://doi.org/10.1007/11787006_1

Fernández JD, Kirrane S, Polleres A, Steyskal S (2017) Self-enforcing access control for encrypted RDF. In: European semantic web conference. Springer, pp 607–622

Fernandez Garcia JD, Kiesling E, Kirrane S, Neuschmid J, Mizerski N, Polleres A, Sabou M, Thurner T, Wetz P (2016) Propelling the potential of enterprise linked data in Austria. Roadmap and report. https://www.linked-data.at/wp-content/uploads/2016/12/propel_book_web.pdf

Gabillon A, Letouzey L (2010) A view based access control model for sparql. In: 2010 4th international conference on network and system security (NSS), pp 105–112

Gavriloaie R, Nejdl W, Olmedilla D, Seamons KE, Winslett M (2004) No registration needed: how to use declarative policies and negotiation to access sensitive resources on the semantic web. In: ESWS. Springer, pp 342–356

Gerbracht S (2008) Possibilities to encrypt an RDF-graph. In: Proceeding of information and communication technologies: from theory to applications, pp 1–6

Giereth M (2005) On partial encryption of RDF-graphs. In: Proceeding of international semantic web conference, vol 3729, pp 308–322

Heitmann B, Hermsen F, Decker S (2017) k-RDF-neighbourhood anonymity: combining structural and attribute-based anonymisation for linked data. In: Proceedings of the 5th workshop on society, privacy and the semantic web – policy and technology (PrivOn2017) (PrivOn). http://ceur-ws.org/Vol-1951/#paper-03

Iannella R, Villata S (2018) ODRL information model 2.2. W3C proposed recommendation, W3C. Available at https://www.w3.org/TR/odrl-model/

Jain A, Farkas C (2006) Secure resource description framework: an access control model. In: Proceedings of the eleventh ACM symposium on access control models and technologies, SACMAT '06. ACM, pp 121–129. http://doi.acm.org/10.1145/1133058.1133076

Kasten A, Scherp A, Armknecht F, Krause M (2013) Towards search on encrypted graph data. In: Proceeding of the international conference on society, privacy and the semantic web-policy and technology, pp 46–57

Kasten A, Scherp A, Schauß P (2014) A framework for iterative signing of graph data on the web. Springer International Publishing, Cham, pp 146–160. https://doi.org/10.1007/978-3-319-07443-6_11.

Kirrane S (2015) Linked data with access control. PhD thesis, INSIGHT Centre for Data Analytics, National University of Ireland, Galway. https://aran.library.nuigalway.ie/handle/10379/4903

Kirrane S, Abdelrahman A, Mileo A, Decker S (2013) Secure manipulation of linked data. In: The semantic web – ISWC 2013. Lecture notes in computer science, vol 8218. Springer, Berlin/Heidelberg, pp 248–263. https://doi.org/10.1007/978-3-642-41335-3_16

Kirrane S, Mileo A, Decker S (2017) Access control and the resource description framework: a survey. Semant Web 8(2):311–352. http://www.semantic-web-journal.net/system/files/swj1280.pdf

Laufer C, Schwabe D (2017) On modeling political systems to support the trust process. In: Proceedings of the 5th workshop on society, privacy and the semantic web – policy and technology (PrivOn2017) (PrivOn). http://ceur-ws.org/Vol-1951/#paper-07

Li N, Li T, Venkatasubramanian S (2007) t-closeness: privacy beyond k-anonymity and l-diversity. In: IEEE 23rd international conference on data engineering, ICDE 2007. IEEE, pp 106–115

Lin Z (2016) From isomorphism-based security for graphs to semantics-preserving security for the resource description framework (RDF). Master's thesis, University of Waterloo

Machanavajjhala A, Gehrke J, Kifer D, Venkitasubramaniam M (2006) l-diversity: privacy beyond k-anonymity. In: Proceedings of the 22nd international conference on data engineering, ICDE'06. IEEE, pp 24–24

Manola F, Miller E (2004) RDF primer. W3C recommendation, W3C. Available at http://www.w3.org/TR/rdf-primer/

Menezes AJ, Van Oorschot PC, Vanstone SA (1996) Handbook of applied cryptography. CRC press, Boca Raton

Radulovic F, García Castro R, Gómez-Pérez A (2015) Towards the anonymisation of RDF data. https://doi.org/10.18293/SEKE2015-167

Reddivari P, Finin T, Joshi A (2005) Policy-based access control for an RDF store. In: Proceedings of the policy management for the web workshop, pp 78–83

Sacco O, Passant A (2011) A privacy preference ontology (PPO) for linked data. In: Linked data on the web, CEUR-WS. http://ceur-ws.org/Vol-813/ldow2011-paper01.pdf

Samarati P, Sweeney L (1998) Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, SRI International

Sandhu RS, Samarati P (1994) Access control: principle and practice. IEEE Commun Mag 32(9):40–48

Seaborne A, Prud'hommeaux E (2008) SPARQL query language for RDF. W3C recommendation, W3C. Available at http://www.w3.org/TR/rdf-sparql-query/

Silva RRC, Leal BC, Brito FT, Vidal VMP, Machado JC (2017) A differentially private approach for querying RDF data of social networks. In: Proceedings of the 21st international database engineering & applications symposium, IDEAS 2017. ACM, New York, pp 74–81. http://doi.acm.org/10.1145/3105831.3105838

Sporny M, Inkster T, Story H, Harbulot B, Bachmann-Gmr R (2011) WebID 1.0 – web identification and discovery. W3C working draft, W3C. Available at http://www.w3.org/2005/Incubator/webid/spec/

Steyskal S, Kirrane S (2015) If you can't enforce it, contract it: enforceability in policy-driven (linked) data markets. In: SEMANTiCS (posters & demos), pp 63–66

Steyskal S, Polleres A (2014) Defining expressive access policies for linked data using the odrl ontology 2.0. In: Proceedings of the 10th international conference on semantic systems. ACM, pp 20–23

Villata S, Delaforge N, Gandon F, Gyrard A (2011) An access control model for linked data. In: On the move

to meaningful internet systems: OTM 2011 workshops, pp 454–463
W3C (n.d.) Webaccesscontrol. Available at https://www.w3.org/wiki/WebAccessControl. Accessed 13 Jan 2018

Security and Privacy for the Resource Description Framework

▸ Security and Privacy Aspects of Semantic Data

Security and Privacy in Big Data Environment

Shekha Chenthara, Hua Wang, and Khandakar Ahmed
Institute for Sustainable Industries and Liveable Cities, VU Research, Victoria University, Melbourne, Australia

Synonyms

Attribute-based access control (ABAC); Authentication, authorization, and accounting (AAA); Availability; Confidentiality; Distributed denial of service (DDoS); Electronic health data (EHD); Electronic health records (EHR); Integrity; Privacy, Authentication, Integrity, Nonrepudiation (PAIN); Ramp secret sharing scheme (RSSS); Role-based access control (RBAC)

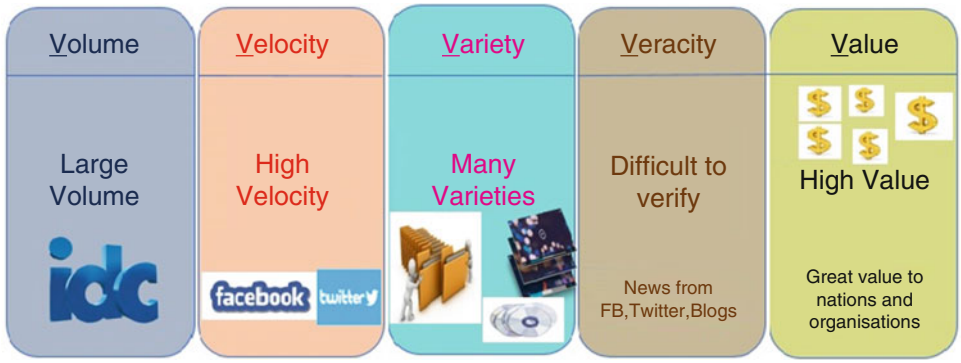
Format Definition

We come across data in every possible form, whether through social media sites, sensor networks, digital images or videos, cell phone GPS signals, purchase transaction records, weblogs, medical records, archives, military surveillance, e-commerce, complex scientific research, and numerous fields. This amount could reach to some quintillion bytes of data! This data is what we call . . . BIG DATA! (Venkatram and Geetha 2017). "Big data is nothing but an assortment of huge and complex data that it becomes very tedious to capture, store, process, retrieve and analyze with the help of on-hand database management tools or traditional data processing techniques." We live in the Age of Big Data where everything that surrounds us is connected to a data source and everything is captured digitally (Matturdi et al. 2014). In the past few years, the total amount of data created by human has been exploded (McCune 1998). From 2005 to 2020, the amount of data is predicted to increase 300 times, from 130 exabytes to 40,000 exabytes (Gantz and Reinsel 2012). These data are generated by scientific research, finance and business informatics, government, Internet search, social networks, document, photography, audio, video, logs, click streams, mobile phones, sensor networks, and so on, and Big Data is the result of this dramatic increase of data.

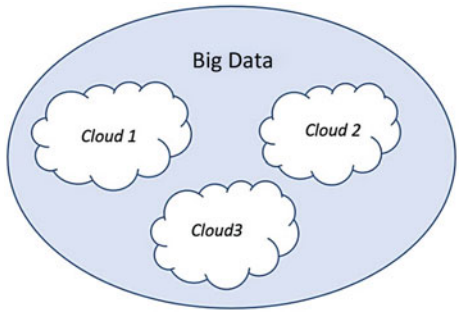
Overview of Big Data

Big Data can be represented by 5Vs – volume, velocity, variety, veracity, and value (Kadhiwala 2017). It is defined by five characteristics of Big Data, including five levels of meaning. Initially, volume refers to huge amount of data. Secondly, velocity refers to fast processing speed. Thirdly, variety refers to different varieties of data categories. Big Data is taken from multiple data sources, and data types contain structured, semi-structured, and unstructured data, such as network log, video, pictures, geographic location information, and so on. Fourthly, veracity refers to the authenticity of data. Finally, value refers to the worth of Big Data (Fang et al. 2017) (Fig. 1).

As the amount of data is increasing day by day, *cloud* has become a perfect solution to store data by providing virtually unlimited storage that can be accessed over the Internet. By outsourcing large volume of data to cloud storage, such as Google Drive, Dropbox, and Amazon Simple Storage Service, users can simplify their data management and reduce data maintenance costs through the pay-as-you-use model (Shu et al. 2018). Cloud services provide the necessary infrastructure by reducing the cost of storing, processing, and updating information with improved



Security and Privacy in Big Data Environment, Fig. 1 5 V features of Big Data



Security and Privacy in Big Data Environment, Fig. 2 An overview of Big Data

efficiency and quality. However, privacy of Big Data is a major hurdle while outsourcing private data in third-party *cloud* as there is a possibility of leaking/sharing sensitive information contained in Big Data with unauthorized entities. As most of the data is sensitive and strictly confidential, security of stored data is a major concern in Big Data environment. Since Big Data often contains sensitive information that needs to be protected from unauthorized access, therefore release of several techniques has to be devised immediately to protect it (Fig. 2).

The content of the chapter is enumerated as follows: Section "Format Definition" describes an overall view of big data. Section "Historical Background" describes challenges and issues of big data security and privacy. Section "Need for Data Security" reveals the need for big data security via some real examples. Section "Key Applications" describes one of the applications related to big data security and privacy and some solutions to enforce big data security and privacy. Section "Future Research Directions" briefly introduces future research areas, and finally, Section "Conclusion" concludes this chapter.

Historical Background

The term "Big Data" is used to indicate the exponential growth and availability, the variety of data, and the speed at which the data is produced and transferred. The rise of Big Data contributes enormous opportunities for individuals, organizations, and society (Huang et al. 2017). An important notion is privacy for Big Data, since it contains sensitive information about individuals. Several privacy models, such as k-anonymity (Sweeney 2002), l-diversity (Machanavajjhala et al. 2007), t-closeness (Li et al. 2007), and differential privacy (Dwork 2011), can be used to anonymize data. It also raises some privacy and ethical issues. Big Data brings some challenges such as heterogeneity, data life cycle management, data processing, scalability, data visualization, security and privacy, etc.

Challenges of Big Data Security and Privacy

Undoubtedly the most challenging and concerned problem in Big Data is security and privacy. Governmental agencies, healthcare industry, biomedical researchers, and private businesses invest enormous resources into the collection, aggre-

gation, and sharing of large amounts of personal data for the tremendous benefit of Big Data. Many facts [section 3] show that Big Data will harm the user's privacy if it is not properly handled (Matturdi et al. 2014). The security and privacy issues which should be concerned in Big Data context include: "1. The personal information of a person when combined with external large data sets, leads to the inference of new facts about that person whereby these facts about the person are sometimes secretive and the person might not want the data owner to know or any person to know about them; 2. Information regarding the users (people) is collected and exploited in order to add value to the business of any organization. This is done by creating insights into their lives which they are unaware of; 3. Another consequence is of Social stratification where a literate person would be taking advantages of the Big data predictive analysis whereas the illiterate/ underprivileged will be worse off it is high in developing countries where 'Digital Divide' is very much prevalent; 4. Big Data used by law enforcement will increase the chances of certain tagged people to suffer from adverse consequences who never have the ability to defend nor have the knowledge of being discriminated against" (Katal et al. 2013). The field of privacy in Big Data which contains a host of challenges involves interaction with individuals, re-identification attacks, probable and provable results, and economic effects (Jensen 2013; Zhang et al. 2014).

Privacy and Security Issues of Big Data

Data is the most crucial part in *Big Data*. Hence, assuring data security and privacy during either data transition or data storage is the core need of Big Data (Kadhiwala 2017; Zhang et al. 2017). Generally, data is treated as secure, when *confidentiality, integrity, and availability*, i.e., the CIA triad model, are satisfied (Xu and Shi 2016). *PAIN* is another measure to ensure data security and privacy benchmark (Sudarsan et al. 2015; Sun et al. 2011a). Hence, favoring literature studies, the main security and privacy issues of Big Data are confidentiality, integrity, availability, monitoring and auditing, key management, and data privacy (Cheng et al. 2017).

Data Confidentiality:Research Directions

Confidentiality relates to applying some rules and restrictions to data against illegal disclosure. Confidentiality can be assured by limiting access to the data and also by employing various cryptographic techniques. There are three different ways to ensure confidentiality in typical security mechanism as follows (Sudarsan et al. 2015):

- Data is encrypted during transition and stored as plaintext.
- Authentication is used on stored plaintext data to grant access.
- Data is encrypted when stored and decrypted when in use.

Several data confidentiality techniques exist, and the most significant techniques among those are access control and encryption. Both techniques have been widely investigated (Bertino et al. 2011; Nabeel et al. 2013; Shang et al. 2010), and both are needed to assure data confidentiality. Confidentiality can also be assured using authentication, one of the AAA security concepts (Li et al. 2011; Ulusoy et al. 2014). Authentication is referred as user identity establishment, whereas authorization is used to grant resource access to the authenticated user. Access control refers to enforcing resource access permission for authorized use to authenticated users. Several access control mechanisms such as mandatory access control (MAC) (Balamurugan et al. 2015), RBAC (Balamurugan et al. 2015; Wang et al. 2005), and ABAC (Ruj 2014) can be used to ensure data privacy.

Integrity

Data integrity provides protection against altering of data by an unauthorized user in an unauthorized manner. Hardware errors, user errors, software errors, or intruders are main reasons for data integrity issues (Sudarsan et al. 2015). Salami attacks, data diddling attacks, trust relationship attacks, man in the middle attack, and

session hijacking attacks are most well-known attacks on data integrity (Types of Network Attacks against Confidentiality, Integrity and Availability 2017). Integrity can be maintained using data provenance, data trustworthiness, data loss, and data deduplication. Data provenance is related to information about creation process as well as sources of data through which it is transformed. It is the process to check all states of data from initial state to current state. Debugging, security, and trust models are various applications of data provenance (Azmi 2015; Glavic 2014). Without data provenance information, the user never comes to know from where the data came and what and which transformations have applied to data. This affects the value or originality of data (Alguliyev and Imamverdiyev 2014).

Availability

Data availability ensures that data must be available for use whenever authorized users require it. However, the emergence of cloud computing has narrowed down issues of data availability for Big Data due to high uptime of cloud. Denial of service (DoS) attack, DDoS attack, and SYN flood attack are known attacks to breach data availability; therefore, there is a need of revised solutions (Types of Network Attacks against Confidentiality, Integrity and Availability 2017).

Key Management

Key management and key sharing between users, servers, and data centers are emerging security issues for Big Data. Wen et al. (Jeong and Shin 2016) have explained various approaches for key management such as secret sharing, server-aided approach, and encryption with signature. In key management with RSSS, users do not need to maintain any key on their own, but instead, they have to share secrets among multiple servers. Key can be reconstructed using a minimum defined number of secrets using RSSS.

Data Privacy

Data privacy intends to assure personally identifiable information (PII) should not be shared without informed assent of related data owner (Kabir et al. 2012; Sun et al. 2011b). In some cases, even though receiving user's consent to use and share the data, the use of PII could be restrained for a specific reason. For example, to develop effective treatment or medicine, the study of patient's medical record is essential. Hence, PII of the patient must be anonymized to protect privacy. In order to ensure data privacy, several encryption techniques and access control mechanisms can be employed.

Need for Data Security: Looking at the Bigger Picture

With the proliferation of data, cyber attacks and data breaches are increasing exponentially (Shen et al. 2017). The growing spate of incidents due to the proliferation and careless handling of data increases the vulnerability of data leakage. With the advent of *cloud*, the data has become more vulnerable to cyber attacks. With the growth of data and its insensitive management, it has come under various sorts of threats that compromise the privacy and security of nations in general and individuals in particular. Recent happenings across the world revealed the vulnerability of data to threats and greater ramifications that followed. Nowadays, no sector is free from data threats and breach. From US presidential elections (where private server of the presidential candidate was hacked) to the recent attack on UIDAI (Unique Identity Development Authority of India) which allots UIN (unique identity number for its citizens), the threat on Big Data is continuing. The incident that shook the conscience of global cyber community was the Ransomware attack that threw bare computers and systems across more than 150 countries at the mercy of cyber thieves. Though some breaches such as "WikiLeaks" and "Panama Papers" expose various irregularities in public domain, these are viewed as great thefts in the literature of data security which put the privacy and security of nations and individuals at stake. Hence there is an imminent need to develop a comprehensive and full proof mechanism to secure the Big Data from unauthorized intrusions.

Key Applications

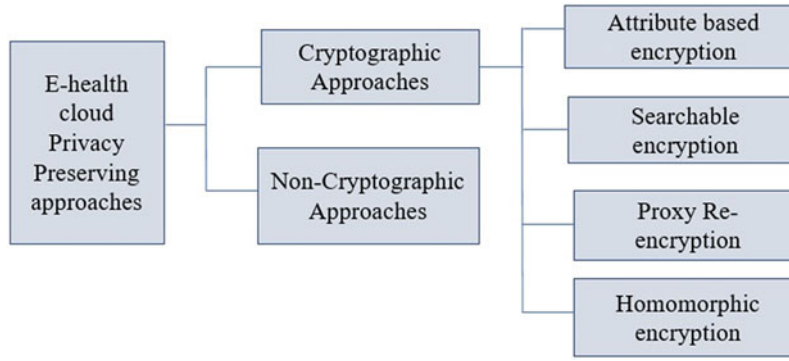
There are diverse privacy and security concerns in various sectors such as social media, banking sectors, healthcare systems, energy industries, and other online database systems. In this study, we discuss one of the research applications related to Big Data in healthcare or EHD that explains how to enforce privacy and security of Big Data.

Application of Privacy and Security in Electronic Health Data (EHD)

EHD (also known as electronic health records, EHR) is a systematic collection of electronic health information about individual patients or populations (Yi et al. 2013). Such records include a whole range of data including demographics, medical histories, medication and allergies, immunization status, laboratory test results, radiology images, billing information, and all sensitive patient information. According to a national survey, 94% of providers report that their EHR makes records readily available at point of care, 88% report that their EHR produces clinical benefits for the practice, and 75% of providers report that their EHR allows them to deliver better patient care (Clemens et al. 2017). However, the transition from paper-based to EHR systems poses some unique challenges for privacy and confidentiality, security, data integrity, and availability. As *cloud computing* is emerging as a new computing paradigm in healthcare sector (Griebel et al. 2015), it not only facilitates the exchange of electronic medical records among healthcare providers or organizations but also acts as a medical record storage center (Li et al. 2016). *Cloud* services provide the infrastructure to healthcare providers and patients by reducing the cost of storing, processing, and updating information with improved efficiency and quality. As computerized medical records are integrated into one place, data can be accessed from different places by different users, and this increases the risk of invasion of privacy. Since most of the data are sensitive and strictly confidential and stored on a third-party server where the owner doesn't have direct access, it demands serious threats in terms of data privacy and security (Abbas and Khan 2014; Vimalachandran et al. 2017). This work focuses on identifying the most appropriate method to share private information between multiple healthcare providers in the patient's care team and the patient and their family or carers in the *cloud* environment.

Hence, EHR in healthcare is facing problems with privacy breaches and unauthenticated record access in the recent years, and the most prime one is related to privacy and security of medical data (Abbas and Khan 2014). The issues range from malware attacks that compromise the integrity of systems and privacy of patients to distributed denial of service (DDoS) which can disrupt facilities' ability to provide patient care. In healthcare systems, for instance, cyber attacks like Ransomware can have ramifications beyond financial loss and breach of privacy (Ahmed and Ullah 2017). Earlier this year, hackers broke into the databases of Community Health Systems (CHS), one of the largest hospital groups in the United States, and accessed personal health information, name, address, and personal data including social security numbers from around 4.5 million patients. Hackers from Internet vigilante group Anonymous also targeted several hospitals, launching a DDoS attack on the hospital website as an act of "hacktivism" (AbuKhousa et al. 2012).

Therefore, there is an indispensable need to protect the privacy, security, confidentiality, integrity, and availability of sensitive information pertaining to individuals' data in general. In this context, cybersecurity is utmost required to prevent, detect, and act on unauthorized access to a health system and its information. Therefore, the primary aim is to strengthen the security infrastructure by providing a strong protection mechanism in "e-healthcare" aiming toward patients' confidence and thereby providing security and privacy (Wu et al. 2012) to electronic health data. Therefore, in order to protect the privacy of patient data, access control mechanisms and encryption techniques will be a better solution. Some of the cryptographic approaches are mentioned in Fig. 3. Narayan et al. (2010) proposed an attribute-based infrastructure for the



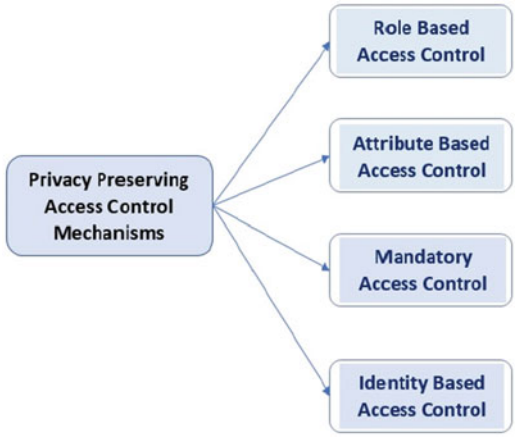
Security and Privacy in Big Data Environment, Fig. 3 Taxonomy of the privacy-preserving approaches in the e-Health cloud

EHR where the patients encrypt their EHR files using the bABE. This approach solves the key management issues by using the users' attribute for data encryption allowing every user to have only one private key for their attribute set for decryption. This approach also allows users to carry and healthcare providers to perform keyword-based searches on the encrypted patients' records without revealing the keywords or partial matches to the cloud system. The keyword search functionality is provided by combining the bABE and the public key encryption with keyword search (PEKS). Ibraimi et al. (2009) proposed a multi-authority scheme for protecting EHD (electronic health data) across different domains. The limitation of the work is that they are not designed for database federations of many medical organizations.

Narayan et al. (2010) proposed a patient-centric cloud-based EHR system by incorporating symmetric key cryptography, public key cryptography, and an attribute-based architecture. This method includes encrypting the patient health data being encrypted by the patient using a symmetric key and also a metadata file which includes a description of the file, attribute-based access policy, and location information encrypted using broadcast CP-ABE and stored in a cloud platform. This method supports direct revocation without re-encryption of data but has a higher computational cost on the patient end where all re-encryption and updating of access policies are handled by the patient side (Narayan et al. 2010). Another downside is that the Trusted Authority can access all the encrypted files. Barua et al. (2013) used the mechanisms of proxy re-encryption and ABE to develop a more sophisticated cloud-based solution. However it does not require external party for key distribution; it leads to a single point of failure and also creates key escrow while managing all attributes with a single authority. In order to resolve the issue, Li et al. (2013) introduced a cloud-based health record sharing scheme using both KP-ABE and MA-ABE schemes. This solution exerts some level of computational cost on the user side.

In EHR systems as most of the data are strictly confidential and stored in a third-party *cloud*, *access control mechanisms* are equally ineludible and vital as *encryption techniques* (Zhang et al. 2015). Access control is a fundamental security barrier for data privacy in a healthcare information system, which limits who can access and operate the documents in an EHR system (Wang et al. 2005, 2009; Zhang et al. 2014). Some access control mechanisms are mentioned in Fig. 4.

Several access control mechanisms such as RBAC and its variants are currently used, and newer approaches such as ABAC have been proposed in healthcare domain to restrict access to electronic health data (Khalil et al. 2007, Wang et al. 2002). However, both of these have few inherent inadequacies as an individual approach. However, by combining the advantages of both RBAC and ABAC, a new approach can be proposed (Alshehri and Raj 2013). This dual-layer access control model being proposed integrates attributes with roles combining the strength of RBAC and ABAC and thereby aims at assuring



Security and Privacy in Big Data Environment, Fig. 4 Basic access control schemes

two fundamental security properties, confidentiality and integrity, of the sensitive data in the healthcare domain. Several research aspects of privacy and security in EHD are categorized and discussed in this study. However, among several encryption techniques, ABE and combination of several access control mechanisms are profoundly important for enforcing security and privacy to EHD.

Future Research Directions

Privacy and security of Big Data is gaining momentum in research community due to emerging technologies like cloud computing, analytics engines, and social networks. There are a number of open problems and future research perspectives related to privacy and security of Big Data (Cuzzocrea 2014, Wang et al. 2015).

1. Privacy-Preserving Big Data Analytics – Big Data are valuable because they are treasured source of knowledge that is useful for decision making and prediction purposes. It possesses challenging research hurdles, because analytics process huge volumes of Big Data, and hence privacy of target data sets is not preserved yet.
2. Privacy-Preserving Social Network Mining – Social network data are the most reliable sources of real-life Big Data, with well-known web social networks like Facebook, Twitter and Instagram. Here, data mining is of primary interest, but the need for privacy and security very often limits the real impact of these tasks.
3. Privacy-Preserving Electronic Health Data – Electronic health data stores sensitive and confidential patient information in large datasets. Hence, there is a high need to preserve the privacy and security of stored data sets to protect the confidentiality of patient.
4. Security Issues of (Big) Outsourced Databases – In cloud infrastructures, databases are often outsourced based on the DaaS (Database as a Service) approach. This elevates more problematic security concern as query-processing procedures may easily access sensitive data sets and determine privacy breaches.

Conclusion

Privacy and security of Big Data is gaining prominence nowadays due to its high proliferation and utility as a result of recent developments in information and communication technology (ICT) such as social networking, IoT (Internet of things), Big Data analytics, and cloud computing. Big Data is a treasure trove of knowledge due to its potential for large-scale use and utility across various fields. This paper discusses scopes, challenges, and various techniques employed to secure privacy and security of Big Data. There is ample scope for further research in Big Data security and privacy of which few areas are discussed under the heading "Future Research Directions" of this write-up. However, in light of existing bottlenecks in data privacy and security, future research directions call for deliberate attention and immense contribution of research scholars and practitioners.

Cross-References

▸ Healthcare
▸ Security and Privacy in Big Data Environment

References

Abbas A, Khan SU (2014) A review on the state-of-the-art privacy-preserving approaches in the e-health clouds. IEEE J Biomed Health Inform 18: 1431–1441

AbuKhousa E, Mohamed N, Al-Jaroodi J (2012) E-health cloud: opportunities and challenges. Futur Internet 4:621–645

Ahmed M, Ullah ASB (2017) False data injection attacks in healthcare. In Australasian Conference on Data Mining. Springer Singapore, Singapore, pp 192–202

Alguliyev R, Imamverdiyev Y (2014) Big data: big promises for information security. In: Proceedings of IEEE 8th international conference on application of information and communication technologies, pp 1–4

Alshehri S, Raj RK (2013) Secure access control for health information sharing systems. In: Proceedings of IEEE international conference on healthcare informatics, pp 277–286

Azmi Z (2015) Opportunities and security challenges of big data. In: Current and emerging trends in cyber operations. Palgrave Macmillan UK, London, pp 181–197

Balamurugan B, Shivitha NG, Monisha V, Saranya V (2015) Survey of access control models for cloud based real-time applications. In: Proceedings of the international conference on innovation information in computing technologies, 2015. IEEE, pp 1–6

Barua M, Lu R, Shen X (2013) SPS: spersonal health information sharing with patient-centric access control in cloud computing. In: Proceedings of the IEEE global communications conference (GLOBECOM), 2013 IEEE. pp 647–652

Bertino E, Ghinita G, Kamra A (2011) Access control for databases: concepts and systems. Found Trends® Databases 3:1–148

Cheng K, Wang L, Shen Y, Wang H, Wang Y, Jiang X, Zhong H (2017) Secure k-NN query on encrypted cloud data with multiple keys. In: IEEE transactions on big data. IEEE. https://doi.org/10.1109/TBDATA.2017.2707552

Clemens S, Alekhya G, Sneha V, Ujwala S, Yazhini C (2017) Impact of electronic health records on long-term care facilities: systematic review. JMIR Med Inform 5:e35. https://doi.org/10.2196/medinform.7958

Cuzzocrea A (2014) Privacy and security of big data: current challenges and future research perspectives. In: Proceedings of the first international workshop on privacy and secuirty of big data. ACM, pp 45–47

Dwork C (2011) Differential privacy. In: Encyclopedia of cryptography and security. Springer-Verlag Berlin, Heidelberg, pp 338–340

Fang W, Wen XZ, Zheng Y, Zhou M (2017) A survey of big data security and privacy preserving. IETE Tech Rev 34:544–560

Gantz J, Reinsel D (2012) The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far east. IDC iView: IDC Analyze Futur 2007: 1–16

Glavic B (2014) Big data provenance: challenges and implications for benchmarking. In: Specifying big data benchmarks. Springer Berlin Heidelberg. pp 72–80

Griebel L et al (2015) A scoping review of cloud computing in healthcare. BMC Med Inform Decis Mak 15:17. https://doi.org/10.1186/s12911-015-0145-7

Huang J, Peng M, Wang H, Cao J, Gao W, Zhang X (2017) A probabilistic method for emerging topic tracking in microblog stream. World Wide Web 20(2): 325–350

Ibraimi L, Asim M, Petković M (2009) Secure management of personal health records by applying attribute-based encryption. In: Proceedings of 2009 6th international workshop on wearable micro and nano technologies for personalized health (pHealth). IEEE, pp 71–74

Jensen M (2013) Challenges of privacy protection in big data analytics. In: Proceedings of 2013 IEEE international congress on big data (BigData Congress). IEEE, pp 235–238

Jeong Y-S, Shin S-S (2016) An efficient authentication scheme to protect user privacy in seamless big data services. Wirel Pers Commun 86:7–19

Kabir ME, Wang H, Bertino E (2012) A role-involved purpose-based access control model. Inf Syst Front 14:809–822

Kadhiwala NJaB (2017) Big data security and privacy issues – a survey. In: Proceedings of the international conference on innovations in power and advanced computing technologies (i-PACT). pp 1–5. https://doi.org/10.1109/IPACT.2017.8245064

Katal A, Wazid M, Goudar R (2013) Big data: issues, challenges, tools and good practices. In: Proceedings of 2013 sixth international conference on contemporary computing (IC3). IEEE, pp 404–409

Khalil F, Wang H, Li J (2007) Integrating markov model with clustering for predicting web page accesses. In: Proceeding of the 13th Australasian world wide web conference, pp 63–74

Li N, Li T, Venkatasubramanian S (2007) t-closeness: privacy beyond k-anonymity and l-diversity. In: Proceeding of the IEEE 23rd international conference on data engineering (ICDE 2007). IEEE, pp 106–115

Li M, Sun X, Wang H, Zhang Y, Zhang J (2011) Privacy-aware access control with trust management in web service. World Wide Web 14:407–430

Li M, Yu S, Zheng Y, Ren K, Lou W (2013) Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. IEEE Trans Parallel Distrib Syst 24:131–143

Li P, Guo S, Miyazaki T, Xie M, Hu J, Zhuang W (2016) Privacy-preserving access to big data in the cloud. IEEE Cloud Comput 3:34–42

Machanavajjhala A, Kifer D, Gehrke J, Venkitasubramaniam M (2007) L-diversity: privacy beyond k-anonymity. ACM Trans Knowl Discov Data 1:3

Matturdi B, Xianwei Z, Shuai L, Fuhong L (2014) Big data security and privacy: a review. China Commun 11:135–145
McCune JC (1998) Data, data, everywhere. Manag Rev 87:10
Nabeel M, Shang N, Bertino E (2013) Privacy preserving policy-based content sharing in public clouds. IEEE Trans Knowl Data Eng 25:2602–2614
Narayan S, Gagné M, Safavi-Naini R (2010) Privacy preserving EHR system using attribute-based infrastructure. In: Proceedings of the 2010 ACM workshop on cloud computing security workshop. ACM, pp 47–52
Ruj S (2014) Attribute based access control in clouds: a survey. In: Proceedings of the 2014 international conference on signal processing and communications (SPCOM). IEEE, pp 1–6
Shang N, Nabeel M, Paci F, Bertino E (2010) A privacy-preserving approach to policy-based content dissemination. In: Proceedings of 2010 IEEE 26th international conference on data engineering (ICDE). IEEE, pp 944–955
Shen Y, Zhang T, Wang Y, Wang H, Jiang X (2017) MicroThings: a generic iot architecture for flexible data aggregation and scalable service cooperation. IEEE Commun Mag 55:86–93
Shu J, Jia X, Yang Kand Wang H (2018) Privacy-preserving task recommendation services for crowdsourcing. IEEE Trans Serv Comput 1(99):1–1
Sudarsan SD, Jetley RP, Ramaswamy S (2015) Security and privacy of big data. In: Big data. Springer India, New Delhi, pp 121–136
Sun X, Wang H, Li J, Pei J (2011a) Publishing anonymous survey rating data. Data Min Knowl Disc 23:379–406
Sun X, Wang H, Li J, Zhang Y (2011b) Injecting purpose and trust into data anonymisation. Comput Secur 30:332–345
Sweeney L (2002) K-anonymity: a model for protecting privacy. Int J Uncertainty Fuzziness Knowledge Based Syst 10:557–570
Types of Network Attacks against Confidentiality, Integrity and Avilability (2017) http://www.omnisecu.com/ccna-security/types-of-network-attacks.php. Accessed 23 Jan 2017
Ulusoy H, Kantarcioglu M, Pattuk E, Hamlen K (2014) Vigiles: fine-grained access control for mapreduce systems. In: Proceedings of 2014 IEEE international congress on big data (BigData Congress). IEEE, pp 40–47
Venkatram K, Geetha MA (2017) Review on big data & analytics – concepts, philosophy, process and applications. Cybern Inf Technol 17:3–27
Vimalachandran P, Wang H, Zhang Y, Zhuo G, Kuang H (2017) Cryptographic access control in electronic health record systems: a security implication. In: Proceedings of the international conference on web information systems engineering. Springer, pp 540–549
Wang H, Cao J, Zhang Y (2002) Ticket-based service access scheme for mobile users. Aust Comput Sci Commun 24(1):285–292
Wang H, Cao J, Zhang Y (2005) A flexible payment scheme and its role-based access control. IEEE Trans Knowl Data Eng 17:425–436
Wang H, Zhang Y, Cao J (2009) Effective collaboration with information sharing in virtual universities. IEEE Trans Knowl Data Eng 21(6):840–853
Wang H, Jiang X, Kambourakis G (2015) Special issue on security, privacy and trust in network-based big data. Inf Sci Int J 318:48–50
Wu R, Ahn G-J, Hu H (2012) Secure sharing of electronic health records in clouds. In: Proceedings of 2012 8th international conference on collaborative computing: networking, applications and worksharing (CollaborateCom). IEEE, pp 711–718
Xu L, Shi W (2016) Security theories and practices for big data. In: Big data concepts, theories, and applications. Springer International Publishing, Cham, pp 157–192
Yi X, Miao Y, Bertino E, Willemson J (2013) Multiparty privacy protection for electronic health records. In: Proceedings of the global communications conference (GLOBECOM), 2013 IEEE. IEEE, pp 2730–2735
Zhang J, Tao X, Wang H (2014) Outlier detection from large distributed databases. World Wide Web 17: 539–568
Zhang Y, Shen Y, Wang H, Yong J, Jiang X (2015) On secure wireless communications for IoT under eavesdropper collusion. IEEE Trans Autom Sci Eng 13(3):1281–1293. July 2016
Zhang J, Li H, Liu X, Luo Y, Chen F, Wang H, Chang L (2017) On efficient and robust anonymization for privacy protection on massive streaming categorical information. IEEE Trans Dependable Secure Comput 14(5):507–520

Self-Supervised Relation Extraction

▸ Distant Supervision from Knowledge Graphs

Semantic Computing

▸ Automated Reasoning

Semantic Data Compression

▸ RDF Compression

Semantic Interlinking

Gianluca Demartini
The University of Queensland, St. Lucia, QLD, Australia

Definitions

Semantic interlinking is defined as the establishment of links and relations between multiple structured datasets.

Overview

Motivation

The exponential growth of data is becoming pervasive across different areas of business and science. Despite its wide availability in large amounts, data is typically stored in standalone silos where different datasets are represented using different formats, stored and indexed within different system architectures, and maintained following different business processes. For example, in certain organizations it is possible to encounter customer databases, technical reports, product images, and other datasets that need to be used in conjunction. Such data integration problems are a long-standing open research challenge in the data management area. The recent rise of big data with its volume and variety dimensions has magnified already existing issues.

Similar challenges are also often present in Open Data where datasets are published and made freely available (typically by governmental organizations) without paying much attention at data quality issues such as the lack of appropriate data format usage and the provision of datasets out of their context.

These situations make data integration an open challenge. *Semantic data interlinking* can be generally defined as the establishment of links and relations between multiple structured datasets. This entry presents an overview of semantic techniques for the interlinking of big data.

Linked Open Data

An important application domain of semantic interlinking is Linked Open Data (LOD) where the goal is to publishing data openly, in a structured format, and interlinked together (Bizer et al. 2007). The aim of the LOD initiative is to fix the challenges that open data comes with, by means of using a common data representation model and by interlinking datasets together.

Tim Berners-Lee suggested a 5-star model for LOD (http://5stardata.info) where he claims that data shared on the Web should ideally be (1) available under an open license, (2) provided as structured data, (3) published using nonproprietary formats, (4) described using unique resource identifiers (URIs), and (5) linked to other data to provide context.

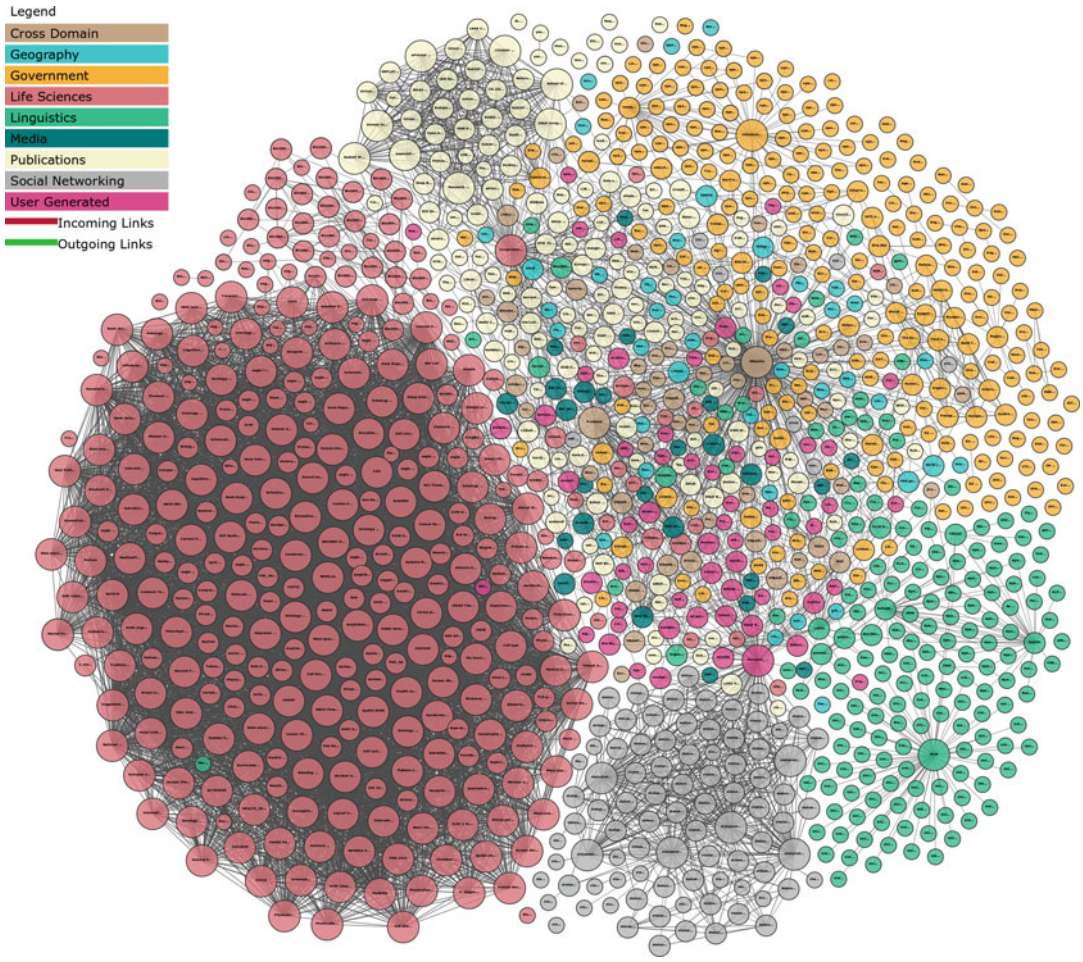
Existing datasets that have been published following such principles are depicted in the LOD cloud (see Fig. 1) where each bubble represents a dataset following LOD principles and edges between nodes represent links across items described in the datasets.

Other than domain-specific datasets (many of which are in the biomedical field), popular datasets in the LOD cloud include DBpedia (Auer et al. 2007) and Wikidata (Vrandečić and Krötzsch 2014).

DBpedia is a structured dataset automatically extracted from Wikipedia infoboxes that contains factual statements about notable entities described in Wikipedia. On the other hand, Wikidata is a crowdsourcing effort aiming at creating a knowledge base of facts. The original motivation to start Wikidata was to supply a central repository for the structured information to be displayed in Wikipedia infoboxes.

Key Research Findings

Research in the area of semantic interlinking focuses on different problems: First, given two datasets to be interlinked, there is the need to align the underlying ontologies used to describe the instances present in the datasets; Next, there is the need to align the instances described in the two datasets, that is, to identify which entities



Semantic Interlinking, Fig. 1 Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/

referred to are the same across the datasets. This section first discusses these two problems and present some methods proposed in the literature designed to address them. It then presents the more specific problem of entity linking, that is, performing semantic interlinking between a document and a structured dataset. Finally, it introduces more recent human-in-the-loop approaches.

Semantic Schema Alignment

The problem of schema alignment (also known as ontology matching Shvaiko and Euzenat 2013) consists in identifying which schema elements of a datasets are equivalent to those of a different dataset. This is a classic problem in the area of semantic interlinking for which a large number of approaches have been proposed.

For example, Jain et al. (2010) focus on methods to find alignments between ontologies used by different LOD datasets. The approach they propose is based on using the Wikipedia category hierarchy to bootstrap the schema alignment process. Similarly, Parundekar et al. (2010) align different ontologies used in different LOD datasets. Their approach is based on existing equivalence statements at the instance level on which they reason about possible schema alignments. To evaluate such methods, standard benchmarks exist. The most popular and commonly used ones

have been created in the context of the Ontology Evaluation Alignment Initiative (Euzenat et al. 2011).

Semantic Record Linkage

Record linkage is defined as the identification of the same real-world entity mentioned across different datasets. In the semantic interlinking domain, this translates into identifying which instances in two LOD datasets refer to the same object.

Approaches for this problem include, for example, Rong et al. (2012) who look at this as a binary classification problem (i.e., a candidate pair of instances matches or not) and solve it using standard supervised machine learning models.

The main challenge of record linkage is scalability: When aiming at identifying duplicates across two datasets, it would be necessary to perform a quadratic number of comparisons to check every possible pair of instances. In a big data context, where the volume of data is prohibitive, performing all possible comparisons is not a scalable option. To deal with this challenge, a number of indexing techniques to make more efficient comparisons have been proposed (Christen 2012). Another common approach to deal with this is *blocking* where the idea is to first use computationally inexpensive methods to create groups of similar objects based on approximate methods (e.g., by means of clustering) and then to perform all possible comparisons using a computationally expensive similarity measure only for the members of a group thus removing a large number of false positives (Bilenko et al. 2006; Papadakis et al. 2013).

Entity Linking

A third problem related to the two main semantic interlinking problems described above is that of *entity linking* (Rao et al. 2013). This is defined as uniquely disambiguating an entity mentioned in a textual document by linking it to a background knowledge graph (Shen et al. 2015). For example, given a document mentioning the entity "Tom Cruise," the goal is to create a link from it to the URI of an instance of a LOD dataset (e.g., DBpedia).

Common approaches for entity linking include, for example, graph-based approaches (Moro et al. 2014) able to look at the coherence of linking decisions based on sub-graph density.

When run at scale, entity linking can benefit from collection features like, for example, entity frequency as an indicator of linking accuracy (Lin et al. 2012). That is, an entity which appears several times in a coherent document collection is likely to be referring to the same concept. It is also easier to use contextual information (e.g., all the sentences where an entity appears) to better decide about which instance of the knowledge graph to link to.

Human-in-the-Loop Semantic Interlinking

A more recent family of approaches to semantic interlinking makes use of the *crowdsourcing* methodology. This implies the development of *human-in-the-loop systems* that leverage machine-based computation to scale interlinking to large datasets but make also use of crowdsourcing to solve difficult cases where humans outperform machine-based algorithms.

An early approach to crowd-based schema alignment was presented by Sarasua et al. (2012) where they compared a human-in-the-loop approach with purely machine-based schema alignment methods showing significant improvements in alignment effectiveness.

For the problem of entity linking, human-in-the-loop solutions include ZenCrowd by Demartini et al. (2012) where authors propose to, given a document and background LOD dataset, collect entity linking decisions from algorithms and from a crowdsourcing platforms. Then, they propose a factor graph model to effectively combine the machine-based and human-based decisions. Again, experimental results show that such combined approach results in better quality entity linking.

A similar human-in-the-loop approach has been proposed by Demartini et al. (2013) for semantic record linkage where the crowdsourcing step is used last after the machine-based blocking and alignment methods described above. For the same problem of semantic record linkage, Wang et al. (2012) investigated the use of grouping

record linkage crowdsourcing tasks together showing how crowd workers perform better as compared to when they are presented with individual record linkage tasks.

Examples of Application

Potential applications where semantic interlinking can provide benefits include information extraction, information retrieval, and knowledge base population.

Applications domains where LOD datasets and semantic interlinking have been used successfully include cultural heritage (Knoblock et al. 2017), manufacturing (Petersen et al. 2017), smart cities (Egami et al. 2016), and others.

Future Directions for Research

As this entry has shown, research in the area of semantic interlinking has been focusing on a number of diverse problems (i.e., semantic schema alignment, semantic record linkage, and entity linking). The current focus of the research community is on the scalability of such approaches to large heterogeneous datasets (e.g., by means of blocking and indexing techniques) and on improving interlinking accuracy by means of human-in-the-loop systems.

Future work should be looking at issues like, for example, semantic interlinking for dynamic LOD datasets (i.e., datasets that are not considered static but rather evolving over time) for which already existing interlinking decisions may need to be updated as ontologies used by the LOD datasets may change (Kuhn et al. 2017).

Cross-References

- ▸ Linked data management
- ▸ Record Linkage
- ▸ Schema Mapping

References

Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives Z (2007) Dbpedia: a nucleus for a web of open data. In: The semantic web, 6th international semantic web conference, 2nd Asian semantic web conference, ISWC 2007 + ASWC 2007, Busan, Korea, 11–15 Nov 2007. Springer, Berlin, pp 722–735

Bilenko M, Kamath B, Mooney RJ (2006) Adaptive blocking: learning to scale up record linkage. In: Sixth international conference on data mining (ICDM'06), pp 87–96. https://doi.org/10.1109/ICDM.2006.13

Bizer C, Heath T, Ayers D, Raimond Y (2007) Interlinking open data on the web. In: Demonstrations track, 4th European semantic web conference, Innsbruck

Christen P (2012) A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans Knowl Data Eng 24(9):1537–1555. https://doi.org/10.1109/TKDE.2011.127

Demartini G, Difallah DE, Cudré-Mauroux P (2012) Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In: Proceedings of the 21st international conference on world wide web. ACM, pp 469–478

Demartini G, Difallah DE, Cudré-Mauroux P (2013) Large-scale linked data integration using probabilistic reasoning and crowdsourcing. VLDB J 22(5):665–687

Egami S, Kawamura T, Ohsuga A (2016) Building urban LOD for solving illegally parked bicycles in Tokyo. In: Proceedings 15th international semantic web conference, part II, the semantic web – ISWC 2016, Kobe, 17–21 Oct 2016, pp 291–307. https://doi.org/10.1007/978-3-319-46547-0_28

Euzenat J, Meilicke C, Stuckenschmidt H, Shvaiko P, Trojahn C (2011) Ontology alignment evaluation initiative: six years of experience. In: Spaccapietra S (ed) Journal on data semantics XV. Springer, Berlin, pp 158–192

Jain P, Hitzler P, Sheth AP, Verma K, Yeh PZ (2010) Ontology alignment for linked open data. Springer, Berlin/Heidelberg, pp 402–417. https://doi.org/10.1007/978-3-642-17746-0_26

Knoblock CA, Szekely PA, Fink EE, Degler D, Newbury D, Sanderson R, Blanch K, Snyder S, Chheda N, Jain N, Krishna RR, Sreekanth NB, Yao Y (2017) Lessons learned in building linked data for the American art collaborative. In: Proceedings of the 16th international semantic web conference, part II, the semantic web – ISWC 2017, Vienna, 21–25 Oct 2017, pp 263–279. https://doi.org/10.1007/978-3-319-68204-4_26

Kuhn T, Willighagen E, Evelo C, Queralt-Rosinach N, Centeno E, Furlong LI (2017) Reliable granular references to changing linked data. Springer International Publishing, Cham, pp 436–451. https://doi.org/10.1007/978-3-319-68288-4_26

Lin T, Mausam, Etzioni O (2012) Entity linking at web scale. In: Proceedings of the joint workshop on automatic knowledge base construction and web-scale knowledge extraction, association for computational linguistics, AKBC-WEKEX '12, Stroudsburg, pp 84–88. http://dl.acm.org/citation.cfm?id=2391200.2391216

Moro A, Raganato A, Navigli R (2014) Entity linking meets word sense disambiguation: a unified approach. Trans Assoc Comput Linguist 2:231–244

Papadakis G, Ioannou E, Palpanas T, Niederee C, Nejdl W (2013) A blocking framework for entity resolution in highly heterogeneous information spaces. IEEE Trans Knowl Data Eng 25(12):2665–2682

Parundekar R, Knoblock CA, Ambite JL (2010) Linking and building ontologies of linked data. Springer, Berlin/Heidelberg, pp 598–614. https://doi.org/10.1007/978-3-642-17746-0_38

Petersen N, Halilaj L, Grangel-González I, Lohmann S, Lange C, Auer S (2017) Realizing an RDF-based information model for a manufacturing company – a case study. In: Proceedings of the 16th international semantic web conference, part II, the semantic web – ISWC 2017, Vienna, 21–25 Oct 2017, pp 350–366. https://doi.org/10.1007/978-3-319-68204-4_31

Rao D, McNamee P, Dredze M (2013) Entity linking: finding extracted entities in a knowledge base. Springer, Berlin/Heidelberg, pp 93–115. https://doi.org/10.1007/978-3-642-28569-1_5

Rong S, Niu X, Xiang EW, Wang H, Yang Q, Yu Y (2012) A machine learning approach for instance matching based on similarity metrics. Springer, Berlin/Heidelberg, pp 460–475. https://doi.org/10.1007/978-3-642-35176-1_29

Sarasua C, Simperl E, Noy NF (2012) Crowdmap: crowdsourcing ontology alignment with microtasks. In: International semantic web conference. Springer, pp 525–541

Shen W, Wang J, Han J (2015) Entity linking with a knowledge base: issues, techniques, and solutions. IEEE Trans Knowl Data Eng 27(2):443–460

Shvaiko P, Euzenat J (2013) Ontology matching: state of the art and future challenges. IEEE Trans Knowl Data Eng 25(1):158–176. https://doi.org/10.1109/TKDE.2011.253

Vrandečić D, Krötzsch M (2014) Wikidata: a free collaborative knowledgebase. Commun ACM 57(10):78–85

Wang J, Kraska T, Franklin MJ, Feng J (2012) Crowder: crowdsourcing entity resolution. Proc VLDB Endow 5(11):1483–1494

Semantic Search

Philippe Cudre-Mauroux
eXascale Infolab, University of Fribourg, Fribourg, Switzerland

Definitions

Semantic Search regroups a set of techniques designed to improve traditional document or knowledge base search. Semantic Search aims at better grasping the context and the semantics of the user query and/or of the indexed content by leveraging natural language processing, Semantic Web, and machine learning techniques to retrieve more relevant results from a search engine.

Overview

Semantic Search is an umbrella term regrouping various techniques for retrieving more relevant content from a search engine. Traditional search techniques focus on ranking documents based on a set of keywords appearing both in the user's query and in the indexed content. Semantic Search, instead, attempts to better grasp the semantics (i.e., meaning) and the context of the user query and/or of the indexed content in order to retrieve more meaningful results.

Semantic Search techniques can be broadly categorized into two main groups depending on the target content:

- techniques improving the relevance of classical search engines where the query consists of natural language text (e.g., a list of keywords) and results are a ranked list of documents (e.g., webpages);
- techniques retrieving semi-structured data (e.g., entities or RDF triples) from a knowledge base (e.g., a knowledge graph or an ontology) given a user query formulated either as natural language text or using a declarative query language like SPARQL.

Those two groups are described in more detail in the following section. For each group, a wide variety of techniques have been proposed, ranging from natural language processing (to better grasp the contents of the query and data) to Semantic Web (to guide the search process leveraging declarative artifacts like ontologies) and machine learning (typically to learn models from large quantities of data).

Main Approaches

We give below an overview of the various techniques that have been proposed in the context of Semantic Search for improving document search

as well as knowledge base search. A number of surveys delve into more detail in this context: Mangold (2007) focuses on natural language queries on RDF knowledge bases or ontologies. Madhu et al. (2011) and Mäkelä (2005) are two brief surveys covering both topics. Bast et al. (2016) is an extensive survey covering Semantic Search in its broadest sense.

Document Search

Classical search engines take as input a user query formulated as a list of keywords and return as output a ranked list of documents relevant to those keywords. A number of Semantic Search methods have been suggested in that context.

Natural language processing (NLP) techniques have long been applied to better grasp the semantics of the query or documents. Often, Part-of-Speech (POS) tagging is first applied on the textual content in order to assign grammatical tags (such as *noun*, *conjunction*, or *verb*) to individual words. Such assignment is highly accurate for well-formed sentences (Manning 2011) but much more challenging for short texts such as queries (Hua et al. 2015). POS tags can then be used to better discriminate textual keywords, for example, for named-entity recognition (NER), where the task is to identify which keywords correspond to real-world entities, or for co-reference resolution, where the task is to identify all keywords referring to the same entity in the text. Sentence parsing takes NLP analyses to the next level by aiming at capturing the overall structure of sentences, typically through a dependency parse tree.

NLP methods are often combined with lexical resources or third-party sources to retrieve more relevant results. The main idea in this context is to identify entities in the textual query or content and to match them to their counterpart in a third-party resource to improve the search results. Voorhees (1993) proposed an early approach in the sense that leverages WordNet to disambiguate word senses and hence improve search results. Pehcevski et al. (2008) analyze the structure of Wikipedia to better rank relevant entities in response to a search request. Kaptein et al. (2010) use Wikipedia to better characterize and identify entities when searching for entities in document collections, while Schuhmacher et al. (2015) combine different features from the documents, the entity mentions, and Wikipedia using a learning-to-rank approach to improve the search results.

Conceptually similar approaches have been proposed in the context of the Semantic Web, by leveraging the structure or contents of a knowledge base to better grasp the context of queries or entities appearing in textual documents. Tran et al. (2007), for instance, propose an ontology-based interpretation of natural language queries for Semantic Search. The authors translate a keyword query into a description logic, conjunctive query that can then be evaluated with respect to an underlying knowledge base. Schuhmacher and Ponzetto (2013) exploit entities and semantic relations from the DBpedia knowledge base to cluster the results of a search engine into more meaningful groups. Prokofyev et al. (2015) leverage an ontology to better resolve co-references in textual documents for Semantic Search tasks.

Machine learning techniques are often used for Semantic Search, to power some of the approaches described above but also to capture the context and semantics of the words or entities appearing in documents. One of the main intuitions in this context is that words that occur in similar contexts are likely to be semantically similar. Early approaches leveraging this observation built high-dimensional matrices capturing the co-occurrence of words in a certain context (e.g., within a window of a few words) and hence the similarity between words (Lund and Burgess 1996). Each word is in that case represented by a sparse vector in a high-dimensional space. Lower-dimensional embeddings can then be created by applying standard matrix factorization techniques like principal component analysis.

More recently, Mikolov et al. (2013) suggested a new word embedding technique to generate dense vector representations of words

efficiently. The method works by maximizing the co-occurrence probability of words appearing within a certain context window using a relatively simple neural network. This opened the door to numerous applications, by efficiently generating vector representations of words from large text corpora and feeding them into subsequent machine learning models. Approaches to improve entity recognition (Siencnik 2015), web search query expansion (Grbovic et al. 2015), or web search ranking (Nalisnick et al. 2016) have, for example, been explored in the context of Semantic Search.

Knowledge Base Search

A number of Semantic Search approaches target large and declarative knowledge bases (a.k.a ontologies or knowledge graphs) instead of document collections. Such knowledge bases can be expressed in many different ways that are typically derived from Semantic Web standards such as RDF or OWL. Google's Knowledge Graph, DBpedia (Bizer et al. 2009), Yago (Rebele et al. 2016), or Wikidata (Vrandecic and Krötzsch 2014) are well-known examples of that trend. Users can express their queries through two main modalities in this case: either as structured (e.g., SPARQL) queries or as natural language (e.g., keyword) queries.

Horrocks and Tessaris (2002) introduced an early formal approach to answer structured queries posed against ontologies. Their algorithm returns sound and complete results to conjunctive queries leveraging reasoning techniques and description logics. Stojanovic et al. (2003) present a method to rank results in ontology-based search. The authors consider conjunctive queries and combine logical inference with an analysis of the contents of the knowledge base to retrieve more relevant results. Maedche et al. (2003) introduce a meta-ontology and a registry to improve search queries targeting ontologies. Their solution leverages WordNet to match entities appearing in the ontology to lexical entries and to guide the search process.

Pound et al. (2010) introduce the ad hoc object retrieval task for searching for resources (e.g., entities, types, or relations) over knowledge bases using natural language queries. The authors also propose a baseline technique for answering such queries based on term frequencies as well as an evaluation methodology. Tonon et al. (2012) propose an improved search technique for ad hoc object retrieval exploiting sequentially an inverted index to answer keyword queries and a graph database to improve the search effectiveness by automatically generating declarative queries over an RDF graph.

Hybrid approaches leveraging both textual and structured contents have also been suggested. Rocha et al. (2004), for instance, combine a traditional search engine with graph exploration techniques to answer keyword queries on an ontology. Zhang et al. (2005) suggest a new model to search semantic portals, where both documents and structured data are available. Their method is based on creating textual representations for all entities in the structured repository such that they can also be indexed and searched through classical information retrieval techniques.

Word embedding techniques (see above) have also been adapted to power Semantic Search on knowledge bases. RDF2Vec (Ristoski and Paulheim 2016), for instance, learns vectorial representations of entities in RDF graphs. Wang et al. (2017) provide a survey of embedding approaches for knowledge bases and classify the models into two main families: translation-based techniques, which interpret relations in the knowledge base as a translation vector between the two entities connected by the relation, and semantic matching models, which exploit similarity-based scoring functions to create the embeddings.

Systems

Leading industrial search engines, such as Bing, Yandex, or Google, all implement Semantic Search in one way or another but typically do not describe in detail the techniques they leverage. A number of Semantic Search systems have been described or open-sourced, however, and are summarized below.

TAP (Guha et al. 2003) is an early Semantic Search framework. TAP focuses on entity search queries expressed as keywords and augments traditional results (documents) with semi-structured data returned from a knowledge base. The knowledge base is also used to better filter and sort the list or returned documents in that context.

A number of Semantic Search systems focusing on RDF and ontologies have been proposed. Swoogle (Ding et al. 2004) offers semantic search over a knowledge base represented in RDF thanks to an inverted index and a database storing metadata about all entities. SWSE (Hogan et al. 2011) follows the typical architecture of a search engine but operates on RDF data also. SWSE results are ranked by running a classical PageRank algorithm on a graph connecting the URIs appearing in the RDF triple to their source on the web.

SemSearch (Lei et al. 2006) is a search engine for the Semantic Web that hides the complexity of the underlying RDF data to the user. SemSearch accepts keyword queries from the user, translates the user queries into formal queries by exploiting the labels of the entities in the knowledge base, runs the resulting query in the knowledge base, and finally ranks the results by taking into account the number of keywords the search results satisfy. Sindice (Oren et al. 2008) is a Semantic Search engine and look-up service that focuses on scaling to very large quantities of semi-structured data. It supports keyword and URI-based search as well as structured queries.

SHOE (Heflin and Hendler 2000) is an early Semantic Search system collecting semi-structured annotations from the web and storing them in a knowledge base. It offers a GUI to formulate ontology-based structured queries to find webpages. The Watson system (d'Aquin and Motta 2011) works similarly by collecting, analyzing, and giving access to ontologies and semantic data available online. It supports both keyword and SPARQL search queries.

SCORE (Sheth et al. 2002) is a platform supporting the creation and maintenance of large knowledge bases. It supports ontology-driven Semantic Search capabilities by extracting facts and metadata from web sources via text mining techniques.

Nordlys (Hasibi et al. 2017) is an open-source toolkit for Semantic Search. The toolkit supports a number of features including detecting entities in natural language queries, cataloging entities from a knowledge base (by default DBpedia), interlinking entities, and retrieving entities.

Schema.org (Mika 2015), finally, is not a system per se but rather a standardization effort founded by leading search engines (including Google, Microsoft, Yahoo, and Yandex). Its mission is to create and promote schemas to embed semi-structured data in web documents (and beyond) in order to facilitate Semantic Search capabilities online.

Cross-References

- Knowledge Graph Embeddings
- Reasoning at Scale
- Semantic Interlinking

References

Bast H, Buchhold B, Haussmann E (2016) Semantic search on text and knowledge bases. Found Trends Inform Retr 10(2–3):119–271. http://dx.doi.org/10.1561/1500000032

Bizer C, Lehmann J, Kobilarov G, Auer S, Becker C, Cyganiak R, Hellmann S (2009) Dbpedia – a crystallization point for the web of data. J Web Sem 7(3):154–165. https://doi.org/10.1016/j.websem.2009.07.002

d'Aquin M, Motta E (2011) Watson, more than a semantic web search engine. Semant Web 2(1):55–63. http://dl.acm.org/citation.cfm?id=2019470.2019476

Ding L, Finin T, Joshi A, Pan R, Cost RS, Peng Y, Reddivari P, Doshi V, Sachs J (2004) Swoogle: a search and metadata engine for the semantic web. In: Proceedings of the thirteenth ACM international conference on information and knowledge management, CIKM'04. ACM, New York, pp 652–659. http://doi.acm.org/10.1145/1031171.1031289

Grbovic M, Djuric N, Radosavljevic V, Silvestri F, Bhamidipati N (2015) Context- and content-aware embeddings for query rewriting in sponsored search. In: Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval, SIGIR'15. ACM, New York, pp 383–392. http://doi.acm.org/10.1145/2766462.2767709

Guha R, McCool R, Miller E (2003) Semantic search. In: Proceedings of the 12th international conference on world wide web, WWW'03. ACM, New York, pp 700–709. http://doi.acm.org/10.1145/775152.775250

Hasibi F, Balog K, Garigliotti D, Zhang S (2017) Nordlys: a toolkit for entity-oriented and semantic search. In: Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval, SIGIR'17. ACM, New York, pp 1289–1292. http://doi.acm.org/10.1145/3077136.3084149

Heflin J, Hendler J (2000) Searching the web with shoe. In: AAAI workshop on artificial intelligence for web search, pp 35–40

Hogan A, Harth A, Umbrich J, Kinsella S, Polleres A, Decker S (2011) Searching and browsing linked data with swse: the semantic web search engine. Web Sem Sci Serv Agents World Wide Web 9(4):365–401. http://www.sciencedirect.com/science/article/pii/S1570826811000473. jWS special issue on Semantic Search

Horrocks I, Tessaris S (2002) Querying the semantic web: a formal approach. In: Horrocks I, Hendler J (eds) The semantic web—ISWC 2002. Springer, Berlin/Heidelberg, pp 177–191

Hua W, Wang Z, Wang H, Zheng K, Zhou X (2015) Short text understanding through lexical-semantic analysis. In: 2015 IEEE 31st international conference on data engineering, pp 495–506. https://doi.org/10.1109/ICDE.2015.7113309

Kaptein R, Serdyukov P, de Vries AP, Kamps J (2010) Entity ranking using wikipedia as a pivot. In: Proceedings of the 19th ACM conference on information and knowledge management, CIKM 2010, Toronto, 26–30 Oct, pp 69–78. http://doi.acm.org/10.1145/1871437.1871451

Lei Y, Uren VS, Motta E (2006) Semsearch: a search engine for the semantic web. In: Proceedings of the 15th international conference on managing knowledge in a world of networks, EKAW 2006, Podebrady, 2–6 Oct 2006, pp 238–245. https://doi.org/10.1007/11891451_22

Lund K, Burgess C (1996) Producing high-dimensional semantic spaces from lexical co-occurrence. Behav Res Methods Instrum Comput 28(2):203–208. https://doi.org/10.3758/BF03204766

Madhu G, Govardhan A, Rajinikanth TV (2011) Intelligent semantic web search engines: a brief survey. CoRR abs/1102.0831. http://arxiv.org/abs/1102.0831, 1102.0831

Maedche A, Motik B, Stojanovic L, Studer R, Volz R (2003) An infrastructure for searching, reusing and evolving distributed ontologies. In: Proceedings of the 12th international conference on world wide web, WWW'03. ACM, New York, pp 439–448. http://doi.acm.org/10.1145/775152.775215

Mäkelä E (2005) Survey of semantic search research. https://seco.cs.aalto.fi/publications/2005/makela-semantic-search-2005.pdf

Mangold C (2007) A survey and classification of semantic search approaches. Int J Metadata Semant Ontologies 2(1):23–34. https://doi.org/10.1504/IJMSO.2007.015073

Manning CD (2011) Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In: Gelbukh AF (ed) Computational linguistics and intelligent text processing. Springer, Berlin/Heidelberg, pp 171–189

Mika P (2015) On schema.org and why it matters for the web. IEEE Int Comput 19(4):52–55. https://doi.org/10.1109/MIC.2015.81

Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. CoRR abs/1301.3781. http://arxiv.org/abs/1301.3781, 1301.3781

Nalisnick E, Mitra B, Craswell N, Caruana R (2016) Improving document ranking with dual word embeddings. In: Proceedings of the 25th international conference companion on world wide web, International world wide web conferences steering committee, WWW'16 Companion. Republic and Canton of Geneva, Switzerland, pp 83–84. https://doi.org/10.1145/2872518.2889361

Oren E, Delbru R, Catasta M, Cyganiak R, Stenzhorn H, Tummarello G (2008) Sindice.com: a document-oriented lookup index for open linked data. IJMSO 3(1):37–52. https://doi.org/10.1504/IJMSO.2008.021204

Pehcevski J, Vercoustre AM, Thom JA (2008) Exploiting locality of wikipedia links in entity ranking. In: Macdonald C, Ounis I, Plachouras V, Ruthven I, White RW (eds) Advances in information retrieval. Springer, Berlin/Heidelberg, pp 258–269

Pound J, Mika P, Zaragoza H (2010) Ad-hoc object retrieval in the web of data. In: Proceedings of the 19th international conference on world wide web, WWW'10. ACM, New York, pp 771–780. http://doi.acm.org/10.1145/1772690.1772769

Prokofyev R, Tonon A, Luggen M, Vouilloz L, Difallah DE, Cudré-Mauroux P (2015) Sanaphor: ontology-based coreference resolution. In: Proceedings of the 14th international conference on the semantic web, ISWC 2015, vol 9366. Springer, Berlin/Heidelberg, pp 458–473. https://doi.org/10.1007/978-3-319-25007-6_27

Rebele T, Suchanek FM, Hoffart J, Biega J, Kuzey E, Weikum G (2016) YAGO: a multilingual knowledge base from wikipedia, wordnet, and geonames. In: Proceedings Part II 15th international semantic web conference of the semantic web, ISWC 2016, Kobe, 17–21 Oct, pp 177–185. https://doi.org/10.1007/978-3-319-46547-0_19

Ristoski P, Paulheim H (2016) Rdf2vec: Rdf graph embeddings for data mining. In: Groth P, Simperl E, Gray A, Sabou M, Krötzsch M, Lecue F, Flöck F, Gil Y (eds) The semantic web – ISWC 2016. Springer International Publishing, Cham, pp 498–514

Rocha C, Schwabe D, Aragao MP (2004) A hybrid approach for searching in the semantic web. In: Proceedings of the 13th international conference on world wide web, WWW'04. ACM, New York, pp 374–383. http://doi.acm.org/10.1145/988672.988723

Schuhmacher M, Ponzetto SP (2013) Exploiting dbpedia for web search results clustering. In: Proceedings of the 2013 workshop on automated knowledge base construction, AKBC'13. ACM, New York, pp 91–96. http://doi.acm.org/10.1145/2509558.2509574

Schuhmacher M, Dietz L, Paolo Ponzetto S (2015) Ranking entities for web queries through text and knowledge. In: Proceedings of the 24th ACM international on conference on information and knowledge management, CIKM'15. ACM, New York, pp 1461–1470. http://doi.acm.org/10.1145/2806416.2806480

Sheth A, Bertram C, Avant D, Hammond B, Kochut K, Warke Y (2002) Managing semantic content for the web. IEEE Int Comput 6(4):80–87. https://doi.org/10.1109/MIC.2002.1020330

Siencnik SK (2015) Adapting word2vec to named entity recognition. In: Proceedings of the 20th Nordic conference of computational linguistics, NODALIDA 2015, 11–13 May. Institute of the Lithuanian Language, Vilnius, pp 239–243. http://aclweb.org/anthology/W/W15/W15-1830.pdf

Stojanovic N, Studer R, Stojanovic L (2003) An approach for the ranking of query results in the semantic web. In: Fensel D, Sycara K, Mylopoulos J (eds) The semantic web – ISWC 2003. Springer, Berlin/Heidelberg, pp 500–516

Tonon A, Demartini G, Cudré-Mauroux P (2012) Combining inverted indices and structured search for ad-hoc object retrieval. In: Proceedings of the 35th international ACM SIGIR conference on research and development in information retrieval, SIGIR'12. ACM, New York, pp 125–134. http://doi.acm.org/10.1145/2348283.2348304

Tran T, Cimiano P, Rudolph S, Studer R (2007) Ontology-based interpretation of keywords for semantic search. In: Proceedings of the 6th international the semantic web and 2nd asian conference on asian semantic web conference, ISWC'07/ASWC'07. Springer, Berlin/Heidelberg, pp 523–536. http://dl.acm.org/citation.cfm?id=1785162.1785201

Voorhees EM (1993) Using wordnet to disambiguate word senses for text retrieval. In: Proceedings of the 16th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR'93. ACM, New York, pp 171–180. http://doi.acm.org/10.1145/160688.160715

Vrandecic D, Krötzsch M (2014) Wikidata: a free collaborative knowledgebase. Commun ACM 57(10):78–85. http://doi.acm.org/10.1145/2629489

Wang Q, Mao Z, Wang B, Guo L (2017) Knowledge graph embedding: a survey of approaches and applications. IEEE Trans Knowl Data Eng 29(12):2724–2743. https://doi.org/10.1109/TKDE.2017.2754499

Zhang L, Yu Y, Zhou J, Lin C, Yang Y (2005) An enhanced model for searching in semantic portals. In: Proceedings of the 14th international conference on world wide web, WWW'05. ACM, New York, pp 453–462. http://doi.acm.org/10.1145/1060745.1060812

Semantic Stream Processing

Danh Le-Phuoc[1] and Manfred Hauswirth[1,2]
[1]Open Distributed Systems, Technical University of Berlin, Berlin, Germany
[2]Fraunhofer FOKUS, Berlin, Germany

Synonyms

RDF stream processing; Stream reasoning

Definitions

Semantic stream processing (SSP) refers to a set of models, principles, and techniques for analyzing and processing stream data by exploiting semantic structures which are explicitly or implicitly embedded in stream data elements. Such "semantic streams" are represented as sequences of temporal graphs linking human-machine understandable semantics and computational primitives. Semantic stream processing approaches leverage reasoning capabilities through formally defined rules to automate and optimize their continuous processing flows formulated in high-level abstract concepts and relationships.

Overview

Billions of sensors being distributed across the globe are continuously streaming data about the physical world around us. The stream data generated by networks of sensors enables us to detect and identify a multitude of things, from simple phenomena to complex events and situations. However, the lack of integration and communication between these networks and the lack of contextual information and background knowledge often isolate important data streams and intensify the existing problems of too much data and not enough knowledge about implicit meaning of such data and user intentions. As a potential remedy for such problems, Whitehouse et al. (2006) proposed the concept of "semantic streams" that

represent stream data associated with logic rules. Such logic rules represented in Prolog allow users to pose declarative queries over semantic interpretations of sensor data. Interestingly, the Prolog-based query model of Whitehouse et al. (2006) from the wireless sensor network community is adopted itself from the semantic Web services paradigm. This paradigm is similar to the concept of "semantic sensor Web" Sheth et al. (2008a) from the semantic Web community. Sheth et al. (2008a) claimed that sensors annotated with semantic metadata will increase interoperability and provide contextual information which is essential for situational knowledge.

As a result, sensor data published as semantic data sources (in RDF models) along with dynamic web data sources (social network, web blogs, etc.), which can be viewed as a variant of sensor data, introduced several research challenges in addressing their highly dynamic and low-latency processing nature. This then opened a new research trend in the semantic Web community, called RDF stream processing (RSP) or Linked Data stream processing (Le-Phuoc et al. 2012b), dealing with heterogeneous stream data sources that can be modeled by means of the RDF model. The choice of RDF as the data model, in combination with ontological profiles for representing stream data elements, offers not only interoperability but also well-understood semantics based on Datalog, Description Logics (DLs), and Answer Set Programming (ASP). Following this design choice, several RSP engines were built by extending existing semantic Web software stacks, e.g., triple storage and SPARQL query engines. Consequently, minimizing differences to "static" RDF and "static" SPARQL became the common goal for a majority of approaches in this line of work. Despite a significant amount of work dedicated to modeling stream data in RDF- and SPARQL-like continuous query languages in the last 10 years, none of them can provide a comprehensive, clean ,and sound theoretical foundation along with a robust implementation.

However, amid the disagreement on a unified data model and query language, there are a lot of parallel efforts in building applications and processing engines in this domain. This movement has started a new research trend of trying to realize reasoning features as offered in conventional RDF engines also for streams, called stream reasoning (Margara et al. 2014; Dell'Aglio et al. 2017). This research domain advocates stream reasoning as the common theme for enabling logic entailment profiles (RDFS, OWL, etc.) of RSP engines as well as other kinds of reasoning over RDF streams, e.g., statistical reasoning. There are few attempts toward this direction, however. The stream reasoning research area remains vastly unexplored, both from a theoretical point of view and also from the perspective of systems and tools supporting it. Notably, the implementations often are done before having a sound theoretical foundation.

In our context, a stream may consist of any kind of raw data with the restriction of discrete elements, e.g., we do not consider "normal" video or audio streams, but discrete pieces of such data. The same holds true for any other data which is discrete in nature, e.g., sensor readings. Semantic streams consist of such discrete elements which are semantically annotated. These annotations can be done manually or by way of automatic annotation through data analysis or hybrid approaches, e.g., the stream and its characteristics are described manually, whereas the individual stream elements are annotated by software. An example for this is monitoring of parking spaces with a camera, where the raw data is not interesting but only if a parking space is occupied or not. Thus the data is "lifted," and this process is called semantic lifting which benefits greatly from the broad availability of machine learning tools, e.g., object recognition with convolution neural networks and entity extraction with natural language processing (NLP) tools. In fact, semantic lifting can tap into a wide range of semantic computing pipelines (Sheu et al. 2010) that has been around more than a decade. It is important to question whether it is trivial to seamlessly integrate these two types into a single processing pipeline. The constituting parts of stream reasoning are grounded in well-understood formal semantics and can usually be expressed via straightforward sets of rules. As such, they do not exhibit the complexity and the

opacity of artificial intelligence approaches that are based on machine learning and neural models.

In semantic stream processing, meaning and relationships do not have to be predefined and "hardwired" into data formats and the application program code at design time. Semantic technology enables encoding and extracting meaning separately from stream data elements and stream sources and separately from application code. With the interlinked nature of semantic data associated with stream data, a software agent can directly search topics, concepts, and associations that span a vast number of stream data sources linked with knowledge graphs. This automatic discovery capability fosters the dynamic composability in a semantic stream processing pipeline. Hence, adding, changing, and implementing new relationships or interconnecting sub-stream processing pipelines in a different way can be done on the fly at run-time after deploying the application logic. In this fashion, a semantic stream processing pipeline can be federated across autonomous processing agents which expose their self-describing stream sources and processing capabilities. Via automatic discovery, the software agent does not have to have full a priori knowledge about input data sources to deploy its application logic. For instance, an autonomous vehicle can continuously discover stream data sources available via its current network connections without knowing them or their types in advance. It can then subscribe the corresponding continuous queries to the relevant stream sources, e.g., to get notifications about traffic jams on the roads and junctions nearby its locations without having to hard-code the data sources and queries.

Key Research Findings

The most consolidated group of research findings on semantic stream processing includes contributions from RSP engines and stream reasoners. These contributions aim at enabling the semantic integration of heterogeneous stream data sources with (large) static datasets stored in relational databases or triple stores through declarative continuous queries based on RDF and SPARQL. By adding window operators to SPARQL and extending the RDF model to capture temporal aspects of stream elements, RSP engines such as C-SPARQL (Barbieri et al. 2010b), CQELS (Le-Phuoc et al. 2011) and SPARQL$_{stream}$ (Calbimonte et al. 2010) integrate the features of data stream management systems (DSMSs) and SPARQL engines. For their implementation, different approaches were chosen based on the features or performance criteria of interest. In the top-down approach, query features are broken down for delegation to the underlying processing engines. The extreme case following the top-down approach is rewriting a continuous query in SPARQL form to SQL queries on traditional relational database or relational data stream management systems. This approach is called ontology-based data access (ODBA), and ODBA-based engines (Calbimonte et al. 2010; Kharlamov et al. 2017) use the ontologies as the semantic guidelines for rewriting their queries to a targeted database schema. EP-SPARQL (Anicic et al. 2010) is another variant of this rewriting approach which translates its unified language for event processing and reasoning as logic expressions to Prolog rules. These are then executed in a Prolog engine. In contrast to this, the bottom-up approach builds or reuses physical query operators to construct execution engines which coordinate and control the query execution process. It takes considerably more effort to build query engines of this kind, e.g., CQELS and C-SPARQL. However, it is easier to improve the performance and add more features as the implementation is under full control. For example, C-SPARQL extended reasoning features, and CQELS implemented more efficient data structures to improve processing throughputs.

To offer a high processing throughput, the system implementers need to spend significant engineering efforts on optimizing and tuning physical data structures (Le-Phuoc 2017; Ren et al. 2017) and query executors (Le-Phuoc et al. 2013; Ren et al. 2017; Bazoobandi et al. 2017) tailored to the

RDF graph nature of stream elements. To scale out the processing to multiple processing nodes, generic cloud-based stream platforms are used: For example, CQELS Cloud (Le-Phuoc et al. 2013) uses Apache Storm and Strider (Ren et al. 2017) uses Apache Spark. The scaling strategy of Strider offers a good solution for handling computationally expensive operations like continuous reasoning on RDFS+ profiles.

In the continuous query setting, the incremental reasoning approach is the obvious choice for materializing the logic entailments of the RDF data within a window. With the observation that deletions can be foreseen and are not random, expiration time annotations are associated with all the axioms involved in the materialization, and such information is exploited to identify only the facts to be deleted. Sparkwave (Komazec et al. 2012) makes use of the RETE (Forgy 1982) algorithm to maintain RDFS entailments with sliding windows defined in C-SPARQL whereby RDFS axioms are encoded as RETE rules and organized in a network. A different approach is adopted by INSTANS (Rinne et al. 2016) which adopts SPARQL 1.1 with an extension to its query evaluation model to continuously query data streams. The implementation of INSTANS also relies on the RETE algorithm: Tasks are expressed as networks of queries and compiled in RETE-like structures to evaluate the results. When new facts are added to the system, they are matched against these rules.

When stream reasoning started to take off in the scientific community, various efforts tried to unify the query language primitives to promote a query model with a sound formalization which can generalize existing query languages such as C-SPARQL, CQELS-QL, and EP-SPARQL. For example, Dell'Aglio et al. (2014) targeted unifying query languages for RSP engines and extended the support for CEP (Dell'Aglio et al. 2016). In parallel, Beck et al. (2015) proposed the LARS framework as a unified way to express stream reasoning primitives under ASP foundations. LARS was later implemented in Laser (Bazoobandi et al. 2017). From a Datalog perspective, windowing operators are just a special case of temporal Datalog variants, so Ronca et al. (2018) recently proposed a Datalog-based logic framework for stream reasoning with a comprehensive complexity analysis.

An additional and noteworthy extension to logic-based reasoning is presented in Barbieri et al. (2010a) where the authors focus on inductive stream reasoning. Inductive stream reasoning involves mining of large portions of data and applying statistical and machine learning techniques to extract new knowledge. The authors propose combining inductive reasoning with deductive reasoning to increase accuracy of the inductive reasoning. The technique has been applied and validated on a real scenario derived from social media analysis, showing the accuracy of the reasoning algorithm in this context. In a similar effort, Chen et al. (2017) showed that DL-based reasoning can integrate efficiently with online streaming mining for traffic data.

Benefiting from the semantic Web adoption, Le-Phuoc et al. (2012a) and Arias Fisteus et al. (2014) proposed middleware solutions transforming and enriching unstructured data streams or raw sensory data to RDF streams by reusing existing tools from the semantic Web stack. Together with other trends on exposing IoT streams using ontologies (Barnaghi et al. 2012), the research community is working very actively on defining suitable ontologies for capturing semantics of sensor data and its contextual information. Examples are the W3C Semantic Sensor Network Ontology (Haller et al. 2017) and the Thing Description (Kaebisc and Kamiya 2018) and Time Ontology (Cox and Little 2017). This lays a solid foundation for semantic lifting which can use available tools for detecting semantic events (Rea et al. 2004; Wang et al. 2006; Chang et al. 2015) and extracting semantic relationships, e.g., from video scenes (Baier et al. 2017), to generate semantic streams.

Examples of Application

Internet of Things (IoT) and Smart Cities: IoT and smart cities have an extremely wide application range where the ideas of semantic sen-

sor Web (Sheth et al. 2008b) and semantic sensor stream (Barnaghi et al. 2013) can be applicable for various applications. In smart cities, most of the applications try to process and understand information relevant for the life of people in a city and use it to make the city more efficient, friendly to the environment, etc. Such applications have access to a huge amount of heterogeneous stream sensor sources. For example, traffic events from social streams (Anantharam et al. 2015) and event streams of big cities (Anantharam et al. 2016), traffic streams (Chen et al. 2017; Eiter et al. 2017), and other IoT streams (Puschmann et al. 2017) are extremely dynamic data sources which demand efficient and scalable algorithms for processing data in near-real-time at the semantic level. In this context, the semantic stream processing must provide enough expressiveness to abstract and derive high-level concepts from low-level and time-annotated data. Moreover, smart cities require the large-scale integration of different data types and sources: As an example, traffic information can be retrieved from sensors, through tracking cell phones in GSM cells, as well as from navigation systems, or posts on social networks.

Another interesting concept is **citizen sensing** which is social sensing enabled by mobile sensors and human computing whereby humans and their devices act as "sensors" and share their observations and views using mobile devices and Web 2.0 services (Sheth 2009). Together with live social semantics (Alani et al. 2009) and social media stream (Balduini et al. 2012), the applications of this type requires semantic analysis of social media by extending traditional analysis based on graphs enriching the connections between people and concepts with semantic annotations. One of the goals of the analysis of social media is to capture hidden relations between people and concepts. Some experiments on the use of social media analysis have been reported in Balduini et al. (2013), where the authors focus on the processing of Twitter posts to extract new information, e.g., how the mood changes during the day, which are the trending topics, etc., during large-scale events (London Olympic Games 2012 and Milano Design Week 2013).

Predictive Maintenance: Sensing devices allow industrial applications to actively and constantly monitor machines to predict maintenance cycles. Such applications include a large number of sensor sources with a wide variety of sensing platforms and types of measurements, not to mention large enterprise knowledge bases, that need to be correlated with stream data to enrich such time series data. For instance, in Kharlamov et al. (2016) the data processing task requires to extract, aggregate, and correlate static data about turbine structure, streaming data produced by up to 2,000 sensors installed in different parts of the turbine, and historical operational data of the reference sensors stored in multiple data sources. Similarly, the application scenarios from Siemens Energy (Kharlamov et al. 2017) and the DEBS challenge 2017 (Gulisano et al. 2017) need efficient and scalable semantic stream processing pipelines to detect abnormal behavior in manufacturing machines and provide actionable insights based on the observations in time.

Future Directions for Research

As an increasing number of semantic streams is made available for a new generation of applications, practitioners are building more and more engines supporting SSP or integrate SSP into current stream engines or RDF stores. To comprehensively support all features of current RSP engines, CEP, and stream reasoning engines, a top-down approach would provide the theoretical foundations associated with feasible implementations for a new generation of SSP engines in the years to come. As shown in the surveys of Margara et al. (2014) and Dell'Aglio et al. (2017), a unified data model associated with well-formulated semantic processing primitives is a vital prerequisite for building a generic execution framework for SSP. The analysis of computational complexity of the resulting processing models should be done at a fine-grain level to give effective guidelines for designing and implementing the underlying engines.

The first challenging problem in generalizing the data model for stream elements or atomic events is capturing their order in the most generic way. As shown in Margara et al. (2014) and Dell'Aglio et al. (2017), current time models based on logical timestamps have their own shortcomings in terms of expressivity and practical implementations. A solution for enforcing causal order in both windowing operators and sequence patterns of events has to be seamlessly integrated with soft-orders entailed by semantic rules. This solution has to take into account out-of-order streams/events, distributed settings, and imprecise timestamps and imprecise clocks. Then the uncertainty properties need to be captured along with semantic relationships and concepts, especially, since there are more and more symbolic values/entities generated by machine learning algorithms from discrete noisy data as semantic streams. Uncertainty can be modeled together with other provenance information which can be represented as ontological contextual information.

Together with accommodating traditional computing primitives from CEP, SPARQL, DSMS, etc., interleaving reasoning primitives without violating the decidability of the whole processing pipeline is a challenging research problem in designing a declarative query language or domain-specific languages (DSLs) for SSP. Some proposals (Barbieri et al. 2010a; Chen et al. 2017) have started investigating the use of inductive and/or statistical reasoning. Also, recent developments in natural language understanding and computer vision have paved the way for a new generation of semantic query capabilities over stream data, e.g., social streams and video streams. Specifically, the visual reasoning capabilities as proposed in Johnson et al. (2017) and Jang et al. (2017) enable the integration of visual question answering (VQA) query fragments into a SSP pipeline. Interestingly, there are various potential links among VQA and NLP and semantic Web. For instance, to improve some state-of-the-art techniques of VQA, Baier et al. (2017) model scene descriptions through RDF triples which are used to answer the queries about visual relationships in human language form as specified in the Visual Genome dataset (Krishna et al. 2016). It is worth noting that the data and queries of the Visual Genome can be naturally expressed in RDF graphs and SPARQL queries, respectively. On the other hand, there are plenty of works on question answering in human language form for RDF or knowledge graphs, e.g., Unger et al. (2012) and Bordes et al. (2014). The aforementioned features are quite appealing for developers amid the rise of transfer learning with a plethora of free pretrained models, but the computational complexity implications need to be investigated thoroughly as well, as they are not yet well understood.

Increasing the expressiveness of how to define a query or a processing pipeline on semantic streams will incur further research and engineering challenges as achieving good performance, and scalability will be necessary and the progress in this domain is rather modest at the moment. For improving performance, the research problems are not limited to designing more efficient algorithms for certain types of computation primitives, e.g., incremental reasoning in windowing data collections, but also include the execution settings such as processing load, resource constraints, and parallelization and coordination assumptions. For instance, continuous optimization considering several dynamic processing aspects such as multiple queries, volume of static data, and input rates is rather challenging even with traditional relational data stream processing. On top of that, a common assumption of processing stream data is that the processing state can be completely loaded into main memory. This might be overly optimistic, and standard techniques like storing to disks or shifting the data and processing to other computing nodes should be considered as optimization strategies. This new working assumption is advocated by the edge computing paradigm which matches the distributed nature of a majority of stream applications. Also and in particular, the federated processing model of RSP will bring new challenges for current information systems, such as the Web. Uniform parallel

computing settings like cloud/private clusters and high-performance computing nodes with large numbers of CPUs and GPUs will play an important role for continuously handling intensive workloads triggered by stream data at Web scale. This provides more options and capabilities along with technical and research challenges in realizing them.

Cross-References

- Adaptive Windowing
- Continuous Queries
- Definition of Data Streams
- Introduction to Stream Processing Algorithms
- Stream Processing Languages and Abstractions
- Linked Data Management
- Linked Geospatial Data
- Semantic Interlinking
- Stream Query Optimization
- Stream Window Aggregation Semantics and Optimization
- Streaming Big Spatial Data
- Types of Stream Processing Algorithms

References

Alani H, Szomszor M, Cattuto C, Van den Broeck W, Correndo G, Barrat A (2009) Live social semantics. Springer, Berlin/Heidelberg, pp 698–714. https://doi.org/10.1007/978-3-642-04930-9_44

Anantharam P, Barnaghi P, Thirunarayan K, Sheth A (2015) Extracting city traffic events from social streams. ACM Trans Intell Syst Technol 6(4):43:1–43:27. https://doi.org/10.1145/2717317

Anantharam P, Thirunarayan K, Marupudi S, Sheth A, Banerjee T (2016) Understanding city traffic dynamics utilizing sensor and textual observations. In: Proceedings of the thirtieth AAAI conference on artificial intelligence (AAAI'16). AAAI Press, pp 3793–3799. http://dl.acm.org/citation.cfm?id=3016387.3016438

Anicic D, Fodor P, Rudolph S, Stühmer R, Stojanovic N, Studer R (2010) A rule-based language for complex event processing and reasoning. In: Proceedings of the fourth international conference on web reasoning and rule systems (RR'10). Springer, Berlin/Heidelberg, pp 42–57

Arias Fisteus J, Fernández García N, Sánchez Fernández L, Fuentes-Lorenzo D (2014) Ztreamy. Web Semant 25(C):16–23. https://doi.org/10.1016/j.websem.2013.11.002

Baier S, Ma Y, Tresp V (2017) Improving visual relationship detection using semantic modeling of scene descriptions. In: d'Amato C, Fernández M, Tamma VAM, Lécué F, Cudré-Mauroux P, Sequeda JF, Lange C, Heflin J (eds) The semantic web – ISWC 2017 – proceedings of 16th international semantic web conference, Vienna, 21–25 Oct 2017, Part I. Lecture notes in computer science, vol 10587, pp 53–68. https://doi.org/10.1007/978-3-319-68288-4_4

Balduini M, Celino I, Dell'Aglio D, Della Valle E, Huang Y, Lee T, Kim SH, Tresp V (2012) Bottari: an augmented reality mobile application to deliver personalized and location-based recommendations by continuous analysis of social media streams. Web Semant 16:33–41. https://doi.org/10.1016/j.websem.2012.06.004

Balduini M, Della Valle E, Dell'Aglio D, Tsytsarau M, Palpanas T, Confalonieri C (2013) Social listening of City scale events using the streaming linked data framework. Springer, Berlin/Heidelberg, pp 1–16. https://doi.org/10.1007/978-3-642-41338-4_1

Barbieri D, Braga D, Ceri S, Valle ED, Huang Y, Tresp V, Rettinger A, Wermser H (2010a) Deductive and inductive stream reasoning for semantic social media analytics. IEEE Intell Syst 25(6):32–41. https://doi.org/10.1109/MIS.2010.142

Barbieri DF, Braga D, Ceri S, Grossniklaus M (2010b) An execution environment for C-SPARQL queries. In: EDBT 2010, pp 441–452

Barnaghi P, Wang W, Henson C, Taylor K (2012) Semantics for the internet of things: early progress and back to the future. Int J Semant Web Inf Syst 8(1):1–21. https://doi.org/10.4018/jswis.2012010101

Barnaghi P, Wang W, Dong L, Wang C (2013) A linked-data model for semantic sensor streams. In: 2013 IEEE international conference on green computing and communications and IEEE internet of things and IEEE Cyber, physical and social computing, pp 468–475. https://doi.org/10.1109/GreenCom-iThings-CPSCom.2013.95

Bazoobandi HR, Beck H, Urbani J (2017) Expressive stream reasoning with laser. CoRR abs/1707.08876. http://arxiv.org/abs/1707.08876

Beck H, Dao-Tran M, Eiter T, Fink M (2015) Lars: a logic-based framework for analyzing reasoning over streams. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence (AAAI'15). AAAI Press, pp 1431–1438. http://dl.acm.org/citation.cfm?id=2887007.2887205

Bordes A, Chopra S, Weston J (2014) Question answering with subgraph embeddings. CoRR abs/1406.3676. http://arxiv.org/abs/1406.3676

Calbimonte JP, Corcho O, Gray AJG (2010) Enabling ontology-based access to streaming data sources. In: Proceedings of the 9th international seman-

tic web conference on the semantic web – volume part I (ISWC'10). Springer, Berlin/Heidelberg, pp 96–111

Chang X, Yang Y, Xing EP, Yu YL (2015) Complex event detection using semantic saliency and nearly-isotonic SVM. In: Proceedings of the 32nd international conference on international conference on machine learning, vol 37, JMLR.org (ICML'15), pp 1348–1357. http://dl.acm.org/citation.cfm?id=3045118.3045262

Chen J, Lécué F, Pan JZ, Chen H (2017) Learning from ontology streams with semantic concept drift. In: Sierra C (ed) Proceedings of the twenty-sixth international joint conference on artificial intelligence (IJCAI 2017), Melbourne, 19–25 Aug 2017, ijcai.org, pp 957–963. https://doi.org/10.24963/ijcai.2017/133

Cox S, Little C (2017) Time ontology in owl. https://www.w3.org/TR/owl-time/. Online; Accessed 21 Mar 2018

Dell'Aglio D, Valle ED, Calbimonte J, Corcho Ó (2014) RSP-QL semantics: a unifying query model to explain heterogeneity of RDF stream processing systems. Int J Semant Web Inf Syst 10(4):17–44. https://doi.org/10.4018/ijswis.2014100102

Dell'Aglio D, Dao-Tran M, Calbimonte JP, Le-Phuoc D, Della Valle E (2016) A query model to capture event pattern matching in RDF stream processing query languages. Springer, Cham, pp 145–162. https://doi.org/10.1007/978-3-319-49004-5_10

Dell'Aglio D, Della Valle E, van Harmelen F, Bernstein A (2017) Stream reasoning: a survey and outlook. Data Sci 01(1–2):59–83

Eiter T, Parreira JX, Schneider P (2017) Spatial ontology-mediated query answering over mobility streams. Springer, Cham, pp 219–237. https://doi.org/10.1007/978-3-319-58068-5_14

Forgy CL (1982) Rete: a fast algorithm for the many pattern/many object pattern match problem. Artif Intell 19(1):17–37. https://doi.org/10.1016/0004-3702(82)90020-0. http://www.sciencedirect.com/science/article/pii/0004370282900200

Gulisano V, Jerzak Z, Katerinenko R, Strohbach M, Ziekow H (2017) The DEBS 2017 grand challenge. In: Proceedings of the 11th ACM international conference on distributed and event-based systems (DEBS'17). ACM, New York, pp 271–273. https://doi.org/10.1145/3093742.3096342

Haller A, Janowicz K, Cox S, Phuoc DL, Taylor K, Lefrançoi M (2017) Semantic sensor network ontology, w3c recomendation. https://www.w3.org/TR/vocab-ssn/. Online; Accessed 21 Mar 2018

Jang Y, Song Y, Yu Y, Kim Y, Kim G (2017) TGIF-QA: toward spatio-temporal reasoning in visual question answering. In: 2017 IEEE conference on computer vision and pattern recognition (CVPR 2017), Honolulu, 21–26 July 2017, pp 1359–1367. https://doi.org/10.1109/CVPR.2017.149

Johnson J, Hariharan B, van der Maaten L, Hoffman J, Fei-Fei L, Zitnick CL, Girshick RB (2017) Inferring and executing programs for visual reasoning. In: IEEE international conference on computer vision (ICCV 2017), Venice, 22–29 Oct 2017. IEEE Computer Society, pp 3008–3017. https://doi.org/10.1109/ICCV.2017.325

Kaebisc S, Kamiya T (2018) Web of things (wot) thing description. https://www.w3.org/TR/wot-thing-description/. Online; Accessed 21 Mar 2018

Kharlamov E, Kotidis Y, Mailis T, Neuenstadt C, Nikolaou C, Özçep Ö, Svingos C, Zheleznyakov D, Brandt S, Horrocks I, Ioannidis Y, Lamparter S, Möller R (2016) Towards analytics aware ontology based access to static and streaming data. Springer, Cham, pp 344–362. https://doi.org/10.1007/978-3-319-46547-0_31

Kharlamov E, Mailis T, Mehdi G, Neuenstadt C, Özçep Ö, Roshchin M, Solomakhina N, Soylu A, Svingos C, Brandt S, Giese M, Ioannidis Y, Lamparter S, Möller R, Kotidis Y, Waaler A (2017) Semantic access to streaming and static data at siemens. Web Semant Sci Serv Agents World Wide Web 44(Suppl C):54–74. https://doi.org/10.1016/j.websem.2017.02.001. http://www.sciencedirect.com/science/article/pii/S1570826817300124; Industry and In-use Applications of Semantic Technologies

Komazec S, Cerri D, Fensel D (2012) Sparkwave: continuous schema-enhanced pattern matching over RDF data streams. In: Proceedings of the 6th ACM international conference on distributed event-based systems (DEBS'12). ACM, New York, pp 58–68. https://doi.org/10.1145/2335484.2335491

Krishna R, Zhu Y, Groth O, Johnson J, Hata K, Kravitz J, Chen S, Kalantidis Y, Li LJ, Shamma DA, Bernstein M, Fei-Fei L (2016) Visual genome: connecting language and vision using crowdsourced dense image annotations. https://arxiv.org/abs/1602.07332

Le-Phuoc D (2017) Operator-aware approach for boosting performance in RDF stream processing. Web Semant Sci Serv Agents World Wide Web 42(Suppl C):38–54. https://doi.org/10.1016/j.websem.2016.04.001. http://www.sciencedirect.com/science/article/pii/S1570826816300014

Le-Phuoc D, Dao-Tran M, Parreira JX, Hauswirth M (2011) A native and adaptive approach for unified processing of linked streams and linked data. In: Proceedings of 10th international semantic web conference, pp 370–388

Le-Phuoc D, Nguyen-Mau HQ, Parreira JX, Hauswirth M (2012a) A middleware framework for scalable management of linked streams. Web Semant Sci Serv Agents World Wide Web 16(Suppl C):42–51. https://doi.org/10.1016/j.websem.2012.06.003. http://www.sciencedirect.com/science/article/pii/S1570826812000728; the Semantic Web Challenge 2011

Le-Phuoc D, Xavier Parreira J, Hauswirth M (2012b) Linked stream data processing. Springer, Berlin/Heidelberg, pp 245–289. https://doi.org/10.1007/978-3-642-33158-9_7

Le-Phuoc D, Quoc HNM, Van CL, Hauswirth M (2013) Elastic and scalable processing of linked stream data in the cloud. In: ISWC 2013 (1), pp 280–297. https://doi.org/10.1007/978-3-642-41335-3_18

Margara A, Urbani J, van Harmelen F, Bal H (2014) Streaming the web: reasoning over dynamic data. Web Semant Sci Serv Agents World Wide Web 25(Suppl C):24–44. https://doi.org/10.1016/j.websem.2014.02.001. http://www.sciencedirect.com/science/article/pii/S1570826814000067

Puschmann D, Barnaghi P, Tafazolli R (2017) Adaptive clustering for dynamic IoT data streams. IEEE Internet Things J 4(1):64–74. https://doi.org/10.1109/JIOT.2016.2618909

Rea N, Dahyot R, Kokaram A (2004) Semantic event detection in sports through motion understanding. Springer, Berlin/Heidelberg, pp 88–97. https://doi.org/10.1007/978-3-540-27814-6_14

Ren X, Curé O, Ke L, Lhez J, Belabbess B, Randriamalala T, Zheng Y, Kepeklian G (2017) Strider: an adaptive, inference-enabled distributed RDF stream processing engine. Proc VLDB Endow 10(12):1905–1908. https://doi.org/10.14778/3137765.3137805

Rinne M, Solanki M, Nuutila E (2016) Rfid-based logistics monitoring with semantics-driven event processing. In: Proceedings of the 10th ACM international conference on distributed and event-based systems (DEBS'16). ACM, New York, pp 238–245. https://doi.org/10.1145/2933267.2933300

Ronca A, Kaminski M, Cuenca Grau B, Motik B, Horrocks I. Stream reasoning in temporal datalog. In: Proceedings of the 32nd AAAI conference on artificial intelligence (AAAI 2018). AAAI Press, New Orleans

Sheth A (2009) Citizen sensing, social signals, and enriching human experience. IEEE Internet Comput 13(4):87–92. https://doi.org/10.1109/MIC.2009.77

Sheth A, Henson C, Sahoo SS (2008a) Semantic sensor web. IEEE Internet Comput 12(4):78–83. https://doi.org/10.1109/MIC.2008.87

Sheth AP, Henson CA, Sahoo SS (2008b) Semantic sensor web. IEEE Internet Comput 12(4):78–83

Sheu P, Yu H, Ramamoorthy CV, Joshi AK, Zadeh LA (2010) Semantic computing. Wiley-IEEE Press, New York

Unger C, Bühmann L, Lehmann J, Ngonga Ngomo AC, Gerber D, Cimiano P (2012) Template-based question answering over RDF data. In: Proceedings of the 21st international conference on world wide web (WWW'12). ACM, New York, pp 639–648. https://doi.org/10.1145/2187836.2187923

Wang T, Li J, Diao Q, Hu W, Zhang Y, Dulong C (2006) Semantic event detection using conditional random fields. In: Proceedings of the 2006 conference on computer vision and pattern recognition workshop (CVPRW'06), IEEE Computer Society, Washington, DC, pp 109–114. https://doi.org/10.1109/CVPRW.2006.190

Whitehouse K, Zhao F, Liu J (2006) Semantic streams: a framework for composable semantic interpretation of sensor data. In: Proceedings of the third European conference on wireless sensor networks (EWSN'06). Springer, Berlin/Heidelberg, pp 5–20. https://doi.org/10.1007/11669463_4

Sentiment-Based Privacy Preserving Data Publishing Techniques for Social Networks

▸ Privacy Cube

Similarity Estimation

▸ Similarity Sketching

Similarity Sketching

Rasmus Pagh
Computer Science Department, IT University of Copenhagen, Copenhagen S, Denmark

Synonyms

Distance estimation; Similarity estimation; Similarity summarization

Overview

Similarity between a pair of objects, usually expressed as a *similarity score* in $[0, 1]$, is a key concept when dealing with noisy or uncertain data, as is common in big data applications.

The aim of *similarity sketching* is to estimate similarities in a (high-dimensional) space using fewer computational resources (time and/or storage) than a naïve approach that stores unprocessed objects. This is achieved using a form of lossy compression that produces succinct representations of objects in the space, from which similarities can be estimated. In some spaces, it is more natural to consider *distances* rather than similarities; we will consider both of these measures of proximity in the following.

Definitions

Formally, consider a space X of objects and a function $d : X \times X \to \mathbf{R}_+$. We refer to d as a *distance function* for X. Similarity sketching with respect to (X, d) is done by using a sketching function $c : X \to \{0,1\}^s$ such that for every pair of objects $x_1, x_2 \in X$, the distance $d(x_1, x_2)$ can be approximated from the knowledge of $c(x_1)$ and $c(x_2)$. Most forms of similarity sketching are *randomized* and produce an estimate $\hat{d}(c(x_1), c(x_2))$ such that with probability $1 - \delta$, it holds that:

$$(1-\varepsilon)\, d(x_1, x_2) \le \hat{d}(c(x_1), c(x_2)) \le (1+\varepsilon)\, d(x_1, x_2), \quad (1)$$

where $\varepsilon, \delta > 0$ are user-specified parameters. Often $c(x)$ itself represents an object in X, intuitively one that is close to x, and $\hat{d}$ is just distance function evaluation. In some cases, it is not possible to achieve the kind of *multiplicative* error guarantee as in (1), and it is instead the case that $\hat{d}(c(x_1), c(x_2))$ differs from $d(x_1, x_2)$ by an additive constant in $[-\varepsilon, \varepsilon]$.

If X is finite, we note that the probability that some distance in X is *not* within the interval (1) is at most $\binom{|X|}{2}\delta$. If this probability is strictly less than 1, it is possible to fix a choice of sketching function c such that all distances estimated fulfill (1). In the case where X is a data set of interest, *data-dependent* similarity sketching methods can improve performance when estimating distances in X; we do not describe such methods here – for more information, see Wang et al. (2017).

Key Techniques

Quantization

In signal processing, an object x from a large or infinite space X can be mapped to a nearby object $c(x)$ in a smaller, finite subset of the space, to enable a succinct representation. Such a mapping, referred to as *quantization*, can be used for similarity sketching by estimating $d(x, y)$ simply as $d(c(x), c(y))$. An efficient quantization method for Euclidean and angular distances is *product quantization* by Jégou et al. (2011).

MinHash

An early form of similarity sketching targeted *Jaccard similarity* of sets. The technique, now known as *MinHash*, was introduced in Broder et al. (1997) and Broder (1997). It works as follows: For a random permutation $r : X \to X$, map a set S to the "MinHash value" $h_r(S) = \min_{x \in S} r(x)$, where the minimum is over an arbitrary, fixed ordering of X. In practice it is hard to construct random permutations, so instead one uses a hash function $r : X \to R$ where R is a large range (e.g., the 64-bit integers) ensuring that collisions in S are unlikely.

It can be shown that if sets S_1 and S_2 have Jaccard similarity J, the probability that $h_r(S_1) = h_r(S_2)$ (i.e., that the MinHash values collide) is J. MinHash is a so-called locality-sensitive hash function (LSH) for which collision probability depends on similarity (or distance). Computing h_r independently k times, with different hash functions r, we expect that the number t of collisions is close to kJ with high probability. Thus, we can estimate J well as $\hat{J} = t/k$, with precision that grows with k.

Improvements. MinHash can be improved in several ways. First of all, the space usage can be reduced by a method called *b-bit MinHash*, which consists of storing a b-bit hash signature rather than the MinHash value itself, yielding collision probability $J + (1-J)/2^b$. The larger collision probability can be taken into account to derive an unbiased estimator for J. Ultimately, b-bit MinHash has greater precision at the same space usage compared to MinHash; see Li and König (2011).

The value of b used in practice is often 1 or 2, with space typical usage of $k = 32$ or $k = 64$ bits. Larger values of b are relevant when estimating Jaccard similarities close to 0. For $b = 1$, it is particularly efficient to compute the number of hash collisions, using bitwise exclusive or in conjunction with a `popcnt` that computes the

number of nonzero bits. When the Jaccard similarity of interest is above 0.8, there are methods that are more precise than b-bit MinHash, for example, *Odd Sketch* due to Mitzenmacher et al. (2014).

The time required for computing k MinHash values (or b-bit MinHash values) can be reduced by the so-called *one-permutation* method that works by initially splitting the set S into k parts and then computing a MinHash value for each part as described by Li et al. (2012). Another approach is *bottom-k sampling*, where the k smallest hash values of a single hash function are used to select a sample; see Thorup (2013) and its references. These methods have been further improved by Dahlgaard et al. (2017).

LSHable Distance Measures

Jaccard similarity is an example of an *LSHable* similarity measure, namely, it allows a random hash function r such that the collision probability equals the similarity. In fact, the MinHash method extends to any LSHable similarity measure by simply replacing the hash function r. An important LSHable distance measure is *angular distance* as shown by Charikar (2002). Angular distance, in turn, can be used to estimate *cosine similarity*. For more discussion of LSHable similarity measures, see Chierichetti and Kumar (2015).

For distance measures that allow an LSH, a similar result can be achieved since the probability of an LSH collision $r(x) = r(y)$ is a decreasing function of $d(x, y)$. The highest accuracy is achieved around distances for which the collision probability is not far from $1/2$, say, in the range $[1/3, 2/3]$. Functions of this form, for example, for Hamming distance or Euclidean distance, are usually found in the literature on search data structures using locality-sensitive hashing; see, e.g., Andoni and Indyk (2008) and its references.

We observe that these similarity sketches can be efficiently computed in a *streaming* setting where only a single element of S is considered at a time. More information on approximate computing for stream analytics can be found elsewhere in this volume.

Dimension Reduction and Embeddings

A special case of similarity sketching is *dimension reduction*: mapping objects to a lower-dimensional space, reducing the representation size, while approximately preserving distances. We refer to the article on dimension reduction in this encyclopedia for more details.

We mention a particular form of dimension reduction called *kernel approximations* where distances after the embedding reflect a distance measure (called a *kernel*) on the original space. Such mappings are possible for a wide class of distance measures defined on Euclidean space; see, e.g., Rahimi and Recht (2007).

Some spaces X' can be *embedded* into a space X for which similarity sketching is possible, generally introducing a distortion of distances. For example, Manhattan (or ℓ_1) distances on $[0, 1]^d$ can be embedded into Hamming space by unary encoding of (quantized) coordinate values; see, e.g., Gionis et al. (1999). An embedding implies a similarity sketch on X' obtained by combining the embedding with a similarity sketch for X. Of course, the precision that can be obtained in this way is limited by the precision of the embedding.

Applications

Similarity sketching is used, for example:

- in situations where exact distance computation is not possible for reasons of space (e.g., main memory indexes supporting near neighbor search, where the set of vectors is too large to fit in the memory),
- when exact computation is undesirable for reasons of time (e.g., when considering a large set of candidate very high-dimensional objects in an algorithm for nearest neighbor search),
- as a preprocessing step in machine learning applications where it serves to reduce dimensionality as well as data size.

Since similarity sketching distorts distances, it will generally change the output compared to an exact algorithm and might introduce false positives as well as false negatives in search applications.

Software Libraries

Similarity sketching is often implemented as part of a particular application, and as such there does not seem to be many general-purpose libraries available. For Euclidean distances among a fixed set of points, the QuadSketch C++ library (https://github.com/talwagner/quadsketch) provides state-of-the-art performance with a data-dependent mapping. The DataSketch library (https://github.com/ekzhu/datasketch) provides Python implementations of variants of MinHash.

Cross-References

- ▸ Approximate Computing for Stream Analytics
- ▸ Dimension Reduction
- ▸ Query Processing – kNN
- ▸ Record Linkage

Acknowledgements This work received support from the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013)/ ERC grant agreement no. 614331.

References

Andoni A, Indyk P (2008) Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun ACM 51(1):117–122

Broder AZ (1997) On the resemblance and containment of documents. In: Proceedings of compression and complexity of sequences. IEEE, pp 21–29

Broder AZ, Glassman SC, Manasse MS, Zweig G (1997) Syntactic clustering of the web. Comput Netw ISDN Syst 29(8):1157–1166

Charikar M (2002) Similarity estimation techniques from rounding algorithms. In: Proceedings of symposium on theory of computing (STOC), pp 380–388

Chierichetti F, Kumar R (2015) Lsh-preserving functions and their applications. J ACM 62(5):33

Dahlgaard S, Knudsen MBT, Thorup M (2017) Fast similarity sketching. In: Proceedings of symposium on foundations of computer science (FOCS), pp 663–671

Gionis A, Indyk P, Motwani R (1999) Similarity search in high dimensions via hashing. In: Proceedings of conference on very large databases (VLDB), pp 518–529

Jégou H, Douze M, Schmid C (2011) Product quantization for nearest neighbor search. IEEE Trans Pattern Anal Mach Intell 33(1):117–128

Li P, König AC (2011) Theory and applications of b-bit minwise hashing. Commun ACM 54(8):101–109

Li P, Owen AB, Zhang C (2012) One permutation hashing. In: Advances in neural information processing systems (NIPS), pp 3122–3130

Mitzenmacher M, Pagh R, Pham N (2014) Efficient estimation for high similarities using odd sketches. In: Proceedings of international world wide web conference (WWW), pp 109–118

Rahimi A, Recht B (2007) Random features for large-scale kernel machines. In: Advances in neural information processing systems (NIPS), pp 1177–1184

Thorup M (2013) Bottom-k and priority sampling, set similarity and subset sums with minimal independence. In: Proceedings of symposium on theory of computing (STOC). ACM, pp 371–380

Wang J, Zhang T, Song J, Sebe N, Shen HT (2017) A survey on learning to hash. IEEE Trans Pattern Anal Mach Intell 13(9) https://doi.org/10.1109/TPAMI.2017.2699960

Similarity Summarization

- ▸ Similarity Sketching

Sliding-Window Aggregation Algorithms

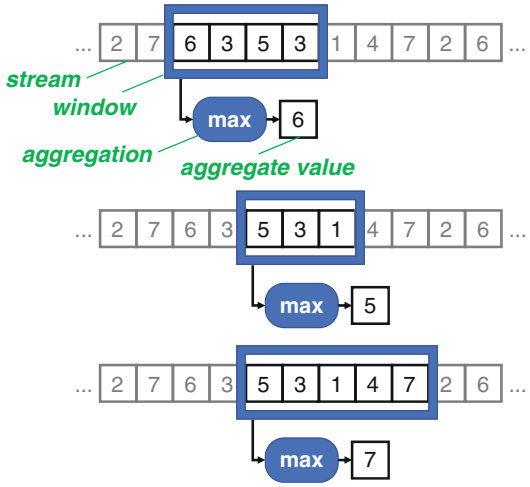
Kanat Tangwongsan[1], Martin Hirzel[2], and Scott Schneider[2]
[1]Mahidol University International College, Salaya, Thailand
[2]IBM Research AI, Yorktown Heights, NY, USA

Synonyms

Sliding-window fold; Stream reduce; SWAG

Definitions

An *aggregation* is a function from a collection of data items to an aggregate value. In *sliding-*



Sliding-Window Aggregation Algorithms, Fig. 1 Sliding-window aggregation definitions

window aggregation, the input collection consists of a window over the most recent data items in a stream. Here, a *stream* is a potentially infinite sequence of data items, and the decision on which data items are *most recent* at any point in time is given by a window policy. A *sliding-window aggregation algorithm* updates the aggregate value, often using incremental-computation techniques, as the window contents change over time, as illustrated in Fig. 1.

Overview

Sliding-window aggregation summarizes a collection of recent streaming data, capturing the most recent happenings as well as some history. Including some history provides context for decisions, which would be missing if only the current data item were used. Using the most recent data helps identify and react to present trends, which would be diluted if all data from the beginning of time were included.

Aggregation is one of the most fundamental data processing operations. This is true in general, not just in stream processing. Aggregation is versatile: it can compute counts, averages, or maxima, index data structures, sketches such as Bloom filters, and many more. In databases, it shows up as a basic relational algebra operator called group-by-aggregate and denoted γ (Garcia-Molina et al. 2008). In spreadsheets, it shows up as a function from a range of cells to a summary statistic (Sajaniemi and Pekkanen 1988). In programming languages, it shows up as a popular higher-order function called `fold` (Hutton 1999). In MapReduce, it shows up as reduce (Dean and Ghemawat 2004), which has been leveraged in many tasks, including computations that do not diminish the volume of data.

In stream processing, aggregation plays a similarly central role. But unlike the abovementioned cases, which focus on data at rest, streaming aggregation must handle data in motion. In particular, sliding-window aggregation must handle inserting new data items into the window as they arrive and evicting old data items from the window as they expire. Supporting this efficiently poses algorithmic challenges, especially for non-invertible aggregation functions such as `max`, for which there is no way to "subtract off" expiring items. From an algorithmic perspective, handling sliding windows with both insertion and eviction is more challenging than handling just insertion. Yet, there are two cases where eviction does not matter: unbounded and tumbling windows. Unbounded windows appear, for instance, in CQL (Arasu et al. 2006). Because they grow indefinitely, it is sufficient to update aggregations upon `insert` and not keep the data item itself around; they never need to call `evict`. Tumbling windows are more common; because they clear the entire contents of the window at the same time, there is no need to call `evict` on individual elements of the window.

Sliding windows are most commonly first-in, first-out (FIFO), resembling the behavior of a queue. What to keep in a sliding window and how often the aggregation is computed are controlled by policies, cataloged elsewhere (Gedik 2013); they may be count-based (e.g., the past 128 elements) or time-based (e.g., the past 12 min), among others. Regardless of policies, FIFO sliding-window aggregation (SWAG) can be formulated as an abstract data type with the following operations:

- `insert`(v) appends the value v to the window.
- `evict`() removes the "oldest" value in the window.
- `query`() returns the aggregation of the values in the window.

Metrics of interest in SWAG implementations are throughput, latency, and memory footprint. SWAG implementations also differ in generality: to enhance efficiency, aggregation operations, when feasible, are applied incrementally – that is, modifying a running sum of sort in response to data items arriving or leaving the window. To what extent this can be exploited depends on the nature of the aggregation operation.

Past work (Gray et al. 1996; Tangwongsan et al. 2015) cast most aggregation operations as binary operators, written $\oplus$, and has categorized them based on algebraic properties. Table 1 lists common aggregation operations with their properties and groups them into categories. An aggregation operator is *invertible* if there exists some function $\ominus$ such that $(x \oplus y) \ominus y = x$ for all x and y. Using $\ominus$, SWAGs can implement eviction as an undo. A function is *associative* if $x \oplus (y \oplus z) = (x \oplus y) \oplus z$ for all x, y, and z. SWAGs can take advantage of associativity by applying $\oplus$ at arbitrary places inside the window. Without associativity, SWAGs are restricted to applying $\oplus$ only at the end, upon insertion. A function is *commutative* if $x \oplus y = y \oplus x$ for all x and y. SWAGs are able to ignore the insertion order of data items for commutative aggregation operators. An aggregation operator is *rank-based* if it relies upon an ordering by some attribute of each data item, for instance, to find the ith-smallest.

Table 2 presents an overview of the SWAG algorithms presented in this article, with their asymptotic complexity, space usage, and restrictions. The most straightforward SWAG algorithm is called **Recalc**, since it always recalculates all values. Upon any `insert` or `evict`, Recalc walks the entire window and recomputes the aggregation value by using all available elements. Its performance is obviously $O(n)$, where n is the current number of elements in the window. Recalc serves as the baseline comparison for all other SWAG algorithms. **Subtract-on-evict** (SOE) is a $O(1)$ algorithm, but it is not general: it can only be used when the aggregation is invertible. Upon every `insert`, SOE updates the current aggregation value using $\oplus$, and upon every `evict`, SOE updates that value using $\ominus$. The **order statistics tree** (OST) adds subtree statistics to the inner nodes of a balanced search tree (Hirzel et al. 2016). Values are put in both a queue and the tree, making both `insert` and `evict` $O(\log n)$. But `query` calls for aggregations such as the median or pth percentile become $O(\log n)$ because such information can be derived by traversing a path that is no longer than the height of the tree.

This section gave a brief overview with background and some simple aggregation algorithms. In general, research into SWAG algorithms tries to avoid $O(n)$ costs (unlike Recalc) but maintain generality (unlike SOE and OST). The next section will discuss the more sophisticated algorithms from Table 2 that offer improvements toward this goal.

Key Research Findings

The most successful techniques in speeding up sliding-window aggregation have been data structuring and algorithmic techniques that yield asymptotic improvements. They are the most effective when the aggregation function meets certain algebraic requirements. For instance, there are important aggregation operations that are associative, but not necessarily invertible nor commutative.

Pre-aggregation of data items that will be evicted at the same time is a technique that can be applied together with all SWAG algorithms discussed in this article. When data items are co-evicted, the window need not store them individually but can instead store partial aggregations, reducing the effective window size n in Table 2. Pre-aggregation algorithms include paned windows (Li et al. 2005), paired windows (Krishnamurthy et al. 2006), and Cutty

Sliding-Window Aggregation Algorithms, Table 1 Aggregation operations. Check marks (✓), crosses (×), and question marks (?) indicate that a property is true for all, false for all, or false for some of the given group, respectively

	Invertible	Associative	Commutative	Rank-based
• **Sum-like**: sum, count, average, standard deviation, ...	✓	✓	✓	×
• **Collect-like**: collect list, concatenate strings, ith-youngest, ...	✓	✓	×	?
• **Median-like**: median, percentile, ith-smallest, ...	✓	✓	✓	✓
• **Max-like**: max, min, argMax, argMin, maxCount, ...	×	✓	?	×
• **Sketch-like**: Bloom filter (Bloom 1970), CountMin (Cormode and Muthukrishnan 2005), HyperLogLog (Flajolet et al. 2007)	×	✓	✓	×

Sliding-Window Aggregation Algorithms, Table 2 Summary of aggregation algorithms and their properties, where n is the window size and $n_{\max}$ is the size of the smallest contiguous range that contains all the shared windows

	Algorithmic complexity		Restrictions
	Time	Space	
Recalc	Worst-case $O(n)$	$O(n)$	None
Subtract-on-Evict (SOE)	Worst-case $O(1)$	$O(n)$	Sum-like or collect-like
Order Statistics Tree (OST) (Hirzel et al. 2016)	Worst-case $O(\log n)$	$O(n)$	Median-like
Reactive Aggregator (RA) (Tangwongsan et al. 2015)	Average $O(\log n)$	$O(n)$	Associative
DABA (Tangwongsan et al. 2017)	Worst-case $O(1)$	$O(n)$	Associative, FIFO
B-Int (Arasu and Widom 2004)	Shared $O(\log n_{\max})$	$O(n_{\max})$	Associative, FIFO
FlatFIT (Shein et al. 2017)	Average $O(1)$	$O(n)$	Associative, FIFO

windows (Carbone et al. 2016). Windows are sometimes coarsened to enable pre-aggregation, improving performance at the expense of some approximation.

B-Int (Arasu and Widom 2004), designed to facilitate sharing across windows, stores a "shared" window S that contains inside it all the windows being shared. To facilitate fast queries, B-Int maintains pre-aggregated values for all base intervals that lie within S. *Base intervals* (more commonly known now as dyadic intervals) are intervals of the form $[2^\ell k, 2^\ell(k+1)-1]$ with $\ell, k \geq 0$. The parameter ℓ defines the level of a base interval. This allows a query between the i-th data item and j-th data item within S to be answered by combining at most $O(\log|i-j|)$ pre-aggregated values, resulting in logarithmic running time.

The **Reactive Aggregator** (RA) (Tangwongsan et al. 2015) is implemented via a balanced tree ordered by time, where internal nodes hold the partial aggregations of their subtrees, and offers $O(\log n)$ amortized time. Instead of the conventional approach to implementing balanced trees by frequent rebalancing, RA projects the tree over a complete perfect binary tree, which it stores in a flat array. This leads to higher performance than other

tree-based SWAG implementations in practice, since it saves the time of rebalancing as well as the overheads of pointers and fine-grained memory allocation.

For latency-sensitive applications, the aggregation algorithm cannot afford a long pause. **DABA** (Tangwongsan et al. 2017) ensures that every SWAG operation takes $O(1)$ time in the worst-case, not just on average. This is accomplished by extending Okasaki's functional queue (Okasaki 1995) and removing dependencies on lazy evaluation and automatic garbage collection. **FlatFIT** (Shein et al. 2017) is another algorithm that achieves $O(1)$ time although in the amortized sense. This is accomplished by storing pre-aggregated values in a tree-like index structure that promotes reuse, reminiscent of path compression in the union-find data structure.

There are a number of other generic techniques that tend to apply broadly to sliding-window aggregation. Window partitioning is sometimes used as a means to maintain group-by aggregation and obtain data parallelism through fission (Schneider et al. 2015). When stream data items arrive out of order, a holding buffer can be used to reorder them before they enter the window (Srivastava and Widom 2004). Alternatively, in the case that the stream is formed by merging multiple sub-streams, out-of-order streams may be solved by pre-aggregating each data source separately and consolidating partial aggregation results as late as possible when doing an actual query (Krishnamurthy et al. 2010).

Examples of Application

Many applications of stream processing depend heavily upon sliding-window aggregation. This section describes concrete examples of applying sliding-window aggregation to real-world use cases. Understanding these examples helps appreciate the problems and guide the design of solutions.

Medical service providers want to save lives by getting early warnings when there is a high likelihood that a patient's health is about to deteriorate. For instance, the Artemis system analyzes data from real-time sensors on patients in a neonatal intensive care unit (Blount et al. 2010). Among other things, it counts how often the blood oxygen saturation and the mean arterial blood pressure fall below a threshold in a 20-s sliding window. If the counts exceed another threshold, Artemis raises an alert.

Financial agents engaged in algorithmic trading want to make money by buying and selling stocks or other financial instruments. Treleaven et al. review the current practice for how that works technologically (Treleaven et al. 2013). Streaming systems for algorithmic trading make their decisions based on predicted future prices. One of the inputs for these predictions is a moving average of the recent history of a price, for example, over a 1-hour sliding window.

Road traffic can be regulated using variable tolling to implement congestion-pricing policies. One of the most popular benchmarks for streaming systems, linear road, is based on variable tolling (Arasu et al. 2004). The idea is to regulate demand by charging higher tolls for driving on congested roads. To do this, the streaming system must determine whether a road is congested. This works by using sliding-window aggregation to compute the number and average speed of vehicles in a given road segment and time window.

The above list of use cases is by no means exhaustive; there are many more applications of sliding-window aggregation, for instance, in phone providers, security, and social media.

Future Directions for Research

Research on sliding-window aggregation has focused mainly on aggregation functions that are associative and on FIFO windows. Much less is known for other nontrivial scenarios. Is it possible to efficiently support associative aggregation functions on windows that are non-FIFO? Besides associativity and invertibility, what other properties can be exploited to develop general-purpose algorithms for fast sliding-window aggregation? How can SWAG algorithms take better advantage of multicore parallelism?

Cross-References

- Adaptive Windowing
- Incremental Sliding Window Analytics
- Stream Query Optimization
- Stream Window Aggregation Semantics and Optimization

References

Arasu A, Widom J (2004) Resource sharing in continuous sliding window aggregates. In: Conference on very large data bases (VLDB), pp 336–347

Arasu A, Cherniack M, Galvez E, Maier D, Maskey AS, Ryvkina E, Stonebraker M, Tibbetts R (2004) Linear road: a stream data management benchmark. In: Conference on very large data bases (VLDB), pp 480–491

Arasu A, Babu S, Widom J (2006) The CQL continuous query language: semantic foundations and query execution. J Very Large Data Bases 15(2):121–142

Bloom BH (1970) Space/time trade-offs in hash coding with allowable errors. Commun ACM 13(7):422–426

Blount M, Ebling MR, Eklund JM, James AG, McGregor C, Percival N, Smith K, Sow D (2010) Real-time analysis for intensive care: development and deployment of the Artemis analytic system. IEEE Eng Med Biol Mag 29:110–118

Carbone P, Traub J, Katsifodimos A, Haridi S, Markl V (2016) Cutty: aggregate sharing for user-defined windows. In: Conference on information and knowledge management (CIKM), pp 1201–1210

Cormode G, Muthukrishnan S (2005) An improved data stream summary: the count-min sketch and its applications. J Algorithms 55(1):58–75

Dean J, Ghemawat S (2004) MapReduce: simplified data processing on large clusters. In: Symposium on operating systems design and implementation (OSDI), pp 137–150

Flajolet P, Fusy E, Gandouet O, Meunier F (2007) HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In: Conference on analysis of algorithms (AofA), pp 127–146

Garcia-Molina H, Ullman JD, Widom J (2008) Database systems: the complete book, 2nd edn. Pearson/Prentice Hall, New Dehli

Gedik B (2013) Generic windowing support for extensible stream processing systems. Softw Pract Exp 44(9): 1105–1128

Gray J, Bosworth A, Layman A, Pirahesh H (1996) Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total. In: International conference on data engineering (ICDE), pp 152–159

Hirzel M, Rabbah R, Suter P, Tardieu O, Vaziri M (2016) Spreadsheets for stream processing with unbounded windows and partitions. In: Conference on distributed event-based systems (DEBS), pp 49–60

Hutton G (1999) A tutorial on the universality and expressiveness of fold. J Funct Program 9(1):355–372

Krishnamurthy S, Wu C, Franklin M (2006) On-the-fly sharing for streamed aggregation. In: International conference on management of data (SIGMOD), pp 623–634

Krishnamurthy S, Franklin MJ, Davis J, Farina D, Golovko P, Li A, Thombre N (2010) Continuous analytics over discontinuous streams. In: International conference on management of data (SIGMOD), pp 1081–1092

Li J, Maier D, Tufte K, Papadimos V, Tucker PA (2005) No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. ACM SIGMOD Rec 34(1):39–44

Okasaki C (1995) Simple and efficient purely functional queues and deques. J Funct Program 5(4): 583–592

Sajaniemi J, Pekkanen J (1988) An empirical analysis of spreadsheet calculation. Softw Pract Exp 18(6):583–596

Schneider S, Hirzel M, Gedik B, Wu KL (2015) Safe data parallelism for general streaming. IEEE Trans Comput 64(2):504–517

Shein AU, Chrysanthis PK, Labrinidis A (2017) FlatFIT: accelerated incremental sliding-window aggregation for real-time analytics. In: Conference on scientific and statistical database management (SSDBM), pp 5:1–5:12

Srivastava U, Widom J (2004) Flexible time management in data stream systems. In: Principles of database systems (PODS), pp 263–274

Tangwongsan K, Hirzel M, Schneider S, Wu KL (2015) General incremental sliding-window aggregation. In: Conference on very large data bases (VLDB), pp 702–713

Tangwongsan K, Hirzel M, Schneider S (2017) Low-latency sliding-window aggregation in worst-case constant time. In: Conference on distributed event-based systems (DEBS), pp 66–77

Treleaven P, Galas M, Lalchand V (2013) Algorithmic trading review. Commun ACM 56(11):76–85

Sliding-Window Fold

- Sliding-Window Aggregation Algorithms

Smart Cities or Future Cities

- Big Data in Smart Cities

Smart City Big Data or Large-Scale Urban Data and/or City Data

▸ Big Data in Smart Cities

Smart Industry

▸ Big Data Warehouses for Smart Industries

SnappyData

Barzan Mozafari
University of Michigan, Ann Arbor, MI, USA

Introduction

An increasing number of enterprise applications, particularly those in financial trading and IoT (Internet of Things), produce mixed workloads with all of the following: (1) continuous stream processing, (2) online transaction processing (OLTP), and (3) online analytical processing (OLAP). These applications need to simultaneously consume high-velocity streams to trigger real-time alerts, ingest them into a write-optimized transactional store, and perform analytics to derive deep insight quickly. Despite a flurry of data management solutions designed for one or two of these tasks, there is no single solution that is apt for all three.

SQL-on-Hadoop solutions (e.g., `Hive`, `Impala/Kudu` and `SparkSQL`) use OLAP-style optimizations and columnar formats to run OLAP queries over massive volumes of static data. While apt for batch processing, these systems are not designed as real-time operational databases, as they lack the ability to mutate data with transactional consistency, to use indexing for efficient point accesses, or to handle high-concurrency and bursty workloads.

Hybrid transaction/analytical processing (HTAP) systems, such as `MemSQL`, support both OLTP and OLAP queries by storing data in dual formats (row and columns) but need to be used alongside an external streaming engine (e.g., `Storm` (Toshniwal et al. 2014), `Kafka`, `Confluent`) to support stream processing. They also lack approximation features required for interactive-speed analytics or visualization workloads (Park et al. 2016).

Finally, there are numerous academic (Chandrasekaran et al. 2003; Mozafari et al. 2012; Thakkar et al. 2011) and commercial (Apache Samza; Toshniwal et al. 2014; TIBCO; Akidau et al. 2013) solutions for stream and event processing. Although some stream processors provide some form of state management or transactions (e.g., `Samza` (Apache Samza), `Liquid` (Fernandez et al. 2015), `S-Store` (Meehan et al. 2015)), they only allow simple queries on streams. However, more complex analytics, such as joining a stream with a large history table, need the same optimizations used in an OLAP engine (Liarou et al. 2012; Braun et al. 2015; Thakkar et al. 2011). For example, streams in IoT are continuously ingested and correlated with large historical data. Trill (Chandramouli et al. 2014) supports diverse analytics on streams and columnar data but lacks transactions. DataFlow (Akidau et al. 2015) focuses on logical abstractions rather than a unified query engine.

Consequently, the demand for mixed workloads has resulted in several composite data architectures, exemplified in the "lambda" architecture, which requires multiple solutions to be stitched together – a difficult exercise that is time-consuming and expensive.

In capital markets, for example, a real-time market surveillance application has to ingest trade streams at very high rates and detect abusive trading patterns (e.g., insider trading). This requires correlating large volumes of data by joining a stream with (1) historical records, (2) other streams, and (3) financial reference

This article is based on Mozafari et al. (2017), authored by Barzan Mozafari, Jags Ramnarayan, Sudhir Menon, Yogesh Mahajan, Soubhik Chakraborty, Hemant Bhanawat, Kishor Bachhav

data which can change throughout the trading day. A triggered alert could in turn result in additional analytical queries, which will need to run on both ingested and historical data. In this scenario, trades arrive on a message bus (e.g., Tibco, IBM MQ, Kafka) and are processed by a stream processor (e.g., Storm) or a homegrown application, while the state is written to a key-value store (e.g., Cassandra) or an in-memory data grid (e.g., GemFire). This data is also stored in HDFS and analyzed periodically using a SQL-on-Hadoop or a traditional OLAP engine.

These heterogeneous workflows, although far too common in practice, have several drawbacks (D1–D4):

D1. Increased complexity and total cost of ownership: The use of incompatible and autonomous systems significantly increases their total cost of ownership. Developers have to master disparate APIs, data models, and tuning options for multiple products. Once in production, operational management is also a nightmare. To diagnose the root cause of a problem, highly paid experts spend hours to correlate error logs across different products.

D2. Lower performance: Performing analytics necessitates data movement between multiple non-colocated clusters, resulting in several network hops and multiple copies of data. Data may also need to be transformed when faced with incompatible data models (e.g., turning Cassandra's ColumnFamilies into Storm's domain objects).

D3. Wasted resources: Duplication of data across different products wastes network bandwidth (due to increased data shuffling), CPU cycles, and memory.

D4. Consistency challenges: The lack of a single data governance model makes it harder to reason about consistency semantics. For instance, a lineage-based recovery in Spark Streaming may replay data from the last checkpoint and ingest it into an external transactional store. With no common knowledge of lineage and the lack of distributed transactions across these two systems, ensuring exactly once semantics is often left as an exercise for the application (Exactly-once processing with trident).

Challenges

SnappyData's goal is to reduce complexity and improve performance by offering streaming, transaction processing, and interactive analytics in a single cluster (Ramnarayan et al. 2016). Realizing this goal involves overcoming several challenges. The first challenge is the drastically different data structures and query processing paradigms that are optimal for each type of workload. For example, column stores are optimal for analytics, transactions need write-optimized row-stores, and infinite streams are best handled by sketches and windowed data structures. Likewise, while analytics thrive with batch processing, transactions rely on point lookups/updates, and streaming engines use delta/incremental query processing. Marrying these conflicting mechanisms in a single system is challenging, as is abstracting away this heterogeneity from programmers.

Another challenge is the difference in expectations of high availability (HA) across different workloads. Scheduling and resource provisioning are also harder in a mixed workload of streaming jobs, long-running analytics, and short-lived transactions. Finally, achieving interactive analytics becomes nontrivial when deriving insight requires joining a stream against large historical data (Makin' Bacon and the Three Main Classes of IoT Analytics).

Approach Overview

To support mixed workloads, SnappyData carefully fuses Apache Spark, as a computational engine, with Apache GemFire, as a transactional store.

Through a common set of abstractions, Spark allows programmers to tackle a confluence of different paradigms (e.g., streaming, machine learning, SQL analytics). Spark's core abstraction, a

Resilient Distributed Dataset (RDD), provides fault tolerance by efficiently storing the lineage of all transformations instead of the data. The data itself is partitioned across nodes and if any partition is lost, it can be reconstructed using its lineage. The benefit of this approach is twofold: avoiding replication over the network and higher throughput by operating on data as a batch. While this approach provides efficiency and fault tolerance, it also requires that an RDD be immutable. In other words, `Spark` is simply designed as a computational framework and therefore (i) does not have its own storage engine and (ii) does not support mutability semantics. (Although IndexedRDD (Indexedrdd for apache spark) offers an updatable key-value store (Indexedrdd for apache spark), it does not support colocation for high-rate ingestions or distributed transactions. It is also unsuitable for HA, as it relies on disk-based checkpoints for fault tolerance.)

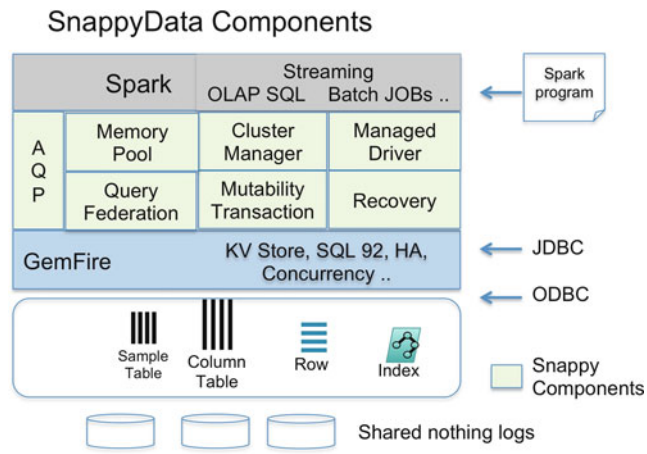
On the other hand, `Apache GemFire` (Apache Geode) (a.k.a. Geode) is one of the most widely adopted in-memory data grids in the industry, which manages records in a partitioned row-oriented store with synchronous replication. It ensures consistency by integrating a dynamic group membership service and a distributed transaction service. `GemFire` allows for indexing and both fine-grained and batched data updates. Updates can be reliably enqueued and asynchronously written back out to an external database. In-memory data can also be persisted to disk using append-only logging with offline compaction for fast disk writes (Apache Geode).

Best of two worlds – To combine the best of both worlds, `SnappyData` seamlessly integrates `Spark` and `GemFire` runtimes, adopting `Spark` as the programming model with extensions to support mutability and HA (high availability) through `GemFire`'s replication and fine-grained updates. This marriage, however, poses several nontrivial challenges. For instance, when ingesting a stream, `SnappyData` processes the incoming stream as a batch, avoids replication, and replays from the source on a failure. To avoid a tuple-at-a-time replication, the processed state can be written to the store in batches. Recovery from failure will thus be limited to the time needed to replay a single batch.

Architecture

Figure 1 depicts `SnappyData`'s core components (the original components from `Spark` and `GemFire` are highlighted).

`SnappyData`'s hybrid storage layer is primarily in-memory and can manage data in row, column, or probabilistic stores. `SnappyData`'s column format is derived from `Spark`'s RDD implementation. `SnappyData`'s row-oriented tables extend `GemFire`'s table and thus support indexing and fast reads/writes on indexed keys. In addition to



SnappyData, Fig. 1 `SnappyData`'s core components

these "exact" stores, SnappyData can also summarize data in *probabilistic* data structures, such as stratified samples and other forms of synopses. SnappyData's query engine has built-in support for approximate query processing (AQP), which can exploit these probabilistic structures. This allows applications to trade accuracy for interactive-speed analytics on streams or massive datasets.

SnappyData supports two programming models – SQL (by extending SparkSQL dialect) and Spark's API. Thus, one can perceive SnappyData as a SQL database that uses Spark's API as its language for stored procedures. Stream processing in SnappyData is primarily through Spark Streaming, but it is modified to run in situ with SnappyData's store.

SQL queries are federated between Spark's Catalyst and GemFire's OLTP engine. An initial query plan determines if the query is a low-latency operation (e.g., a key-based lookup) or a high-latency one (scans/aggregations). SnappyData avoids scheduling overheads for OLTP operations by immediately routing them to appropriate data partitions.

To support replica consistency, fast point updates, and instantaneous detection of failure conditions in the cluster, SnappyData relies on GemFire's P2P (peer-to-peer) cluster membership service (Apache Geode). Transactions follow a two-phase commit protocol using GemFire's Paxos implementation to ensure consensus and view consistency across the cluster.

A Unified API

Spark offers a rich procedural API for querying and transforming disparate data formats (e.g., JSON, Java Objects, CSV). Likewise, to retain a consistent programming style, SnappyData offers its mutability functionalities as extensions of SparkSQL's dialect and its DataFrame API. These extensions are backward compatible, i.e., applications that do not use them observe Spark's original semantics.

A DataFrame in Spark is a distributed collection of data organized into named columns. A DataFrame can be accessed from a SQLContext, which itself is obtained from a SparkContext (a SparkContext is a connection to Spark's cluster). Likewise, much of SnappyData's API is offered through SnappyContext, which is an extension of SQLContext. Listing 1 is an example of using SnappyContext.

```
1 // Create a SnappyContext from a SparkContext
2 val spContext = new org.apache.spark.
      SparkContext(conf)
3 val snpContext = org.apache.spark.sql.
      SnappyContext (spContext)
4
5 // Create a column table using SQL
6 snpContext.sql("CREATE TABLE MyTable (id int,
      data string) using column")
7
8 // Append contents of a DataFrame
      into the table
9 someDataDF.write.insertInto("MyTable");
10
11 // Access the table as a DataFrame
12 val myDataFrame: DataFrame = snpContext.
      table("MyTable")
13 println(s"Number of rows in MyTable = {
      myDataFrame.count()}")
```

Listing 1 Working with DataFrames in SnappyData

Stream processing often involves maintaining counters or more complex multidimensional summaries. As a result, stream processors today are either used alongside a scale-out in-memory key-value store (e.g., Storm with Redis or Cassandra) or come with their own basic form of state management (e.g., Samza, Liquid (Fernandez et al. 2015)). These patterns are often implemented in the application code using simple get/put APIs. While these solutions scale well, most users tend to modify their search patterns and trigger rules quite often. These modifications require expensive code changes and lead to brittle and hard-to-maintain applications.

In contrast, SQL-based stream processors offer a higher-level abstraction to work with streams but primarily depend on row-oriented stores (e.g., IBM; TIBCO; Meehan et al. (2015)) and are thus limited in supporting complex analytics. To support continuous queries with scans, aggregations, top-K queries, and joins with historical and reference data, some of the

same optimizations found in OLAP engines must be incorporated in the streaming engine (Liarou et al. 2012). Thus, SnappyData extends Spark Streaming to allow declaring and querying streams in SQL. More importantly, SnappyData provides OLAP-style optimizations to enable scalable stream analytics, including columnar formats, approximate query processing, and co-partitioning (SnappyData 2016).

Hybrid Store: Row and Column Tables

Tables can be partitioned or replicated and are primarily managed in memory with one or more consistent replicas. The data can be managed in Java heap memory or off-heap. Partitioned tables are always partitioned horizontally across the cluster. For large clusters, SnappyData allows data servers to belong to one or more logical groups, called "server groups." The storage format can be "row" (either partitioned or replicated tables) or "column" (only supported for partitioned tables) format. Row tables incur a higher in-memory footprint but are well suited to random updates and point lookups, especially with in-memory indexes. Column tables manage column data in contiguous blocks and are compressed using dictionary, run-length, or bit encoding (Xin and Rosen).

SnappyData extends Spark's column store to support mutability. Updating row tables is trivial. When records are written to column tables, they first arrive in a *delta row buffer* that is capable of high write rates and then age into a columnar form. The delta row buffer is merely a partitioned row table that uses the same partitioning strategy as its base column table. This buffer table is backed by a conflating queue that periodically empties itself as a new batch into the column table. Here, conflation means that consecutive updates to the same record result in only the final state getting transferred to the column store. For example, inserted/updated records followed by deletes are removed from the queue. The delta row buffer itself uses copy-on-write semantics to ensure that concurrent application updates do not cause inconsistency (Abadi et al. 2013). SnappyData extends Spark's Catalyst optimizer to merge the delta row buffer during query execution.

Probabilistic Store

Achieving interactive response time is challenging when running complex analytics on streams, e.g., joining a stream with a large table (Mozafari and Zaniolo 2010). Even OLAP queries on stored datasets can take tens of seconds to complete if they require a distributed shuffling of records or if hundreds of concurrent queries run in the cluster (Agarwal et al. 2012). In such cases, SnappyData's storage engine is capable of using probabilistic structures to dramatically reduce the volume of input data and provide approximate but extremely fast answers. In this regard, SnappyData can be seen as the first full-fledged commercial AQP engine (see Mozafari (2017) for why the adoption of AQP has not previously slowed). SnappyData's probabilistic structures include uniform samples, stratified samples, and sketches (Mozafari and Niu 2015). Unlike VerdictDB (Park et al. 2018; He et al. 2018) and Database Learning (Park et al. 2017), which are agnostic of the underlying query engine, SnappyData's probabilistic store is tightly integrated into its query processing logic. SnappyData's approach is also different from other AQP engines (Zeng et al. 2014a; Agarwal et al. 2012; Zeng et al. 2014b), in the way that it creates and maintains these structures efficiently and in a distributed manner. However, given these structures, SnappyData uses off-the-shelf error estimation techniques (Agarwal et al. 2014). SnappyData's sample selection and maintenance strategies are discussed next.

Sample selection – Unlike uniform samples, choosing which stratified samples to build is a nontrivial problem. The key question is which sets of columns to build a stratified sample on. Prior work has used skewness, popularity, and storage cost as the criteria for choosing column sets (Agarwal et al. 2012, 2013). SnappyData extends these criteria as follows: for any declared or foreign-key join, the join key is included in a stratified sample in at least one of the participating relations (tables or streams). However,

SnappyData never includes a table's primary key in its stratified sample(s). Furthermore, SnappyData uses an open-source tool, called WorkloadMiner, which automatically analyzes past query logs and reports a rich set of statistics (CliffGuard). These statistics guide SnappyData's users through the sample selection process. WorkloadMiner is integrated into CliffGuard. CliffGuard guarantees a robust physical design (e.g., set of samples), which remains optimal even if future queries deviate from past ones (Mozafari et al. 2015).

Once a set of samples is chosen, the challenge is how to update them, which is a key differentiator between SnappyData and previous AQP systems that use stratified samples (Chaudhuri et al. 2007; Agarwal et al. 2013; Zeng et al. 2015).

Sample maintenance – Previous AQP engines that use offline sampling update and maintain their samples periodically using a single scan of the entire data (Mozafari and Niu 2015). This strategy is not suitable for SnappyData with streams and mutable tables for two reasons. First, maintaining per-stratum statistics across different nodes in the cluster is a complex process. Second, updating a sample in a streaming fashion requires maintaining a reservoir (Vitter et al. 1985; Al-Kateb and Lee 2010), which means the sample must either fit in memory or be evicted to disk. Keeping samples entirely in memory is impractical for infinite streams unless the sampling rate is perpetually decreased. Likewise, disk-based reservoirs are inefficient as they require retrieving and removing individual tuples from disk as new tuples are sampled.

To solve these problems, SnappyData always includes timestamp as an additional column in every stratified sample. Uniform samples are treated as a special case with only one stratified column, i.e., timestamp. As new tuples arrive in a stream, a new batch (in row format) is created for maintaining a sample of each observed value of the stratified columns. Whenever a batch size exceeds a certain threshold (1M tuples by default), it is evicted and archived to disk (in a columnar format), and a new batch is started for that stratum.

Treating each micro-batch as an independent stratified sample has several benefits. First, this allows SnappyData to adaptively adjust the sampling rate for each micro-batch without the need for internode communications in the cluster. Second, once a micro-batch is completed, its tuples never need to be removed or replaced, and therefore they can be safely stored in a compressed columnar format and even archived to disk. Only the latest micro-batch needs to be in-memory and in row format. Finally, each micro-batch can be routed to a single node, reducing the need for network shuffles.

State Sharing

SnappyData hosts GemFire's tables in the executor nodes as either partitioned or replicated tables. When partitioned, the individual buckets are presented as Spark RDD partitions, and their access is therefore parallelized. This is similar to the way that any external data source is accessed in Spark, except that the common operators are optimized in SnappyData. For example, by keeping each partition in columnar format, SnappyData avoids additional copying and serialization and speeds up scan and aggregation operators. SnappyData can also colocate tables by exposing an appropriate partitioner to Spark.

Native Spark applications can register any DataFrame as a temporary table. In addition to being visible to the Spark application, such a table is also registered in SnappyData's catalog – a shared service that makes tables visible across Spark and GemFire. This allows remote clients connecting through ODBC/JDBC to run SQL queries on Spark's temporary tables as well as tables in GemFire.

In streaming scenarios, the data can be sourced into any table from parent stream RDDs (DStream), which themselves could source events from an external queue, such as Kafka. To minimize shuffling, SnappyData tables can preserve the partitioning scheme used by their parent RDDs. For example, a Kafka queue listening on Telco CDRs (call detail records) can be partitioned on `subscriberID` so that Spark's DStream and the SnappyData table ingesting these records will be partitioned on the same key.

Locality-Aware Partition Design

A major challenge in horizontally partitioned distributed databases is to restrict the number of nodes involved in order to minimize (i) shuffling during query execution and (ii) distributed locks (Helland 2007; Zamanian et al. 2015). In addition to network costs, shuffling can also cause CPU bottlenecks by incurring excessive copying (between kernel and user space) and serialization costs (Ousterhout et al. 2015). To reduce the need for shuffling and distributed locks, SnappyData's data model promotes two fundamental ideas:

1. **Co-partitioning with shared keys** – A common technique in data placement is to take the application's access patterns into account. SnappyData pursues a similar strategy: since joins require a shared key, it co-partitions related tables on the join key. SnappyData's query engine can then optimize its query execution by localizing joins and pruning unnecessary partitions.
2. **Locality through replication** – Star schemas are quite prevalent, wherein a few ever-growing fact tables are related to several dimension tables. Since dimension tables are relatively small and change less often, schema designers can ask SnappyData to replicate these tables. SnappyData particularly uses these replicated tables to optimize joins.

Dynamic rebalancing of data – When access is non-uniformly distributed across the keys, a load imbalance occurs where a few servers end up performing most of the work. For instance, when tracking users' browsing behavior on a website, a few popular pages will dominate the rest. SnappyData provides metrics on which nodes are being accessed heavily and also provides administrative APIs that can be used to move "hot buckets" of data to a different node. If the imbalance is a memory usage imbalance, admin APIs can be used to trigger a rebalance which is a non blocking operation that moves buckets of data to less loaded nodes in the background and restores memory balance. Used effectively, rebalancing prevents hotspots from developing in the system and avoid performance bottlenecks.

Hybrid Cluster Manager

Spark applications run as independent processes in the cluster, coordinated by the application's main program, called the driver program. Spark applications connect to cluster managers (YARN or Mesos) to acquire executor nodes. While Spark's approach is appropriate for long-running tasks, as an operational database, SnappyData's cluster manager must meet additional requirements, such as high concurrency, high availability, and consistency.

High Availability

To ensure high availability (HA), SnappyData needs to detect faults and be able to recover from them instantly.

Failure detection – Spark uses heartbeat communications with a central master process to determine the fate of the workers. Since Spark does not use a consensus-based mechanism for failure detection, it risks shutting down the entire cluster due to master failures. However, as an always-on operational database, SnappyData needs to detect failures faster and more reliably. For faster detection, SnappyData relies on UDP neighbor ping and TCP ack timeout during normal data communications. To establish a new, consistent view of the cluster membership, SnappyData relies on GemFire's weighted quorum-based detection algorithm (Apache Geode). Once GemFire establishes that a member has indeed failed, it ensures that a consistent view of the cluster is applied to all members, including the Spark master, driver, and data nodes.

Failure recovery – Recovery in Spark is based on logging the transformations used to build an RDD (i.e., its lineage) rather than the actual data. If a partition of an RDD is lost, Spark has sufficient information to recompute just that partition (Zaharia et al. 2012). Spark can also checkpoint RDDs to stable storage to shorten

the lineage, thereby shortening the recovery time. The decision of when to checkpoint, however, is left to the user. GemFire, on the other hand, relies on replication for instantaneous recovery but at the cost of lower throughput. SnappyData merges these recovery mechanisms as follows:

1. Fine-grained updates issued by transactions avoid the use of Spark's lineage altogether and instead use GemFire's eager replication for fast recovery.
2. Batched and streaming micro-batch operations are still recovered by RDD's lineage, but instead of HDFS, SnappyData writes their checkpoints to GemFire's in-memory storage, which itself relies on a fast P2P (peer-to-peer) replication for recovery. Also, SnappyData's intimate knowledge of the load on the storage layer, the data size, and the cost of recomputing a lost partition allows for automating the choice of checkpoint intervals based on an application's tolerance for recovery time.

Hybrid Scheduler and Provisioning

Thousands of concurrent clients can simultaneously connect to a SnappyData cluster. To support this degree of concurrency, SnappyData categorizes incoming requests as low- and high-latency operations. By default, SnappyData treats a job as a low-latency operation unless it accesses a columnar table. However, applications can also explicitly label their latency sensitivity. SnappyData allows low-latency operations to bypass Spark's scheduler and directly operate on the data. High-latency operations are passed through Spark's fair scheduler. However, among the low-latency operations, SnappyData still relies on a simple FIFO policy (other systems, such as MariaDB or MySQL, use more sophisticated algorithms for transaction scheduling, e.g., VATS (Huang et al. 2017) or MySQL's CATS (Tian et al. 2018)). For low-latency operations, SnappyData attempts to reuse their executors to maximize their data locality (in-process). For high-latency jobs, SnappyData dynamically expands their compute resources while retaining the nodes caching their data.

Consistency Model

SnappyData relies on GemFire for its consistency model. GemFire supports "read committed" and "repeatable read" transaction isolation levels using a variant of the Paxos algorithm (Gray and Lamport 2006). Transactions detect write-write conflicts and assume that writers rarely conflict. When write locks cannot be obtained, transactions abort without blocking (Apache Geode).

SnappyData extends Spark's `SparkContext` and `SQLContext` to add mutability semantics. SnappyData gives each SQL connection its own `SQLContext` in Spark to allow applications to start, commit, and abort transactions.

While any RDD obtained by a Spark program observes a consistent view of the database, multiple programs can observe different views when transactions interleave. An MVCC mechanism (based on GemFire's internal row versions) can be used to deliver a single snapshot view to the entire application.

In streaming applications, upon faults, Spark recovers lost RDDs from their lineage. This means that some subset of the data will be replayed. To cope with such cases, SnappyData ensures the exactly once semantics at the storage layer so that multiple write attempts are idempotent, hence relieving developers of having to ensure this in their own applications. SnappyData achieves this goal by placing the entire flow as a single transactional unit of work, whereby the source (e.g., a Kafka queue) is acknowledged only when the micro-batch is entirely consumed and the application state is successfully updated. This ensures automatic rollback of incomplete transactions.

Conclusion

SnappyData is a unified platform for real-time operational analytics, which supports OLTP, OLAP, and stream analytics in a single integrated solution. SnappyData's approach is a deep integration of a computational engine for high-throughput analytics (Spark) with a scale-out in-memory transactional store (GemFire). SnappyData extends SparkSQL and Spark Streaming APIs

with mutability semantics and offers various optimizations to enable colocated processing of streams and stored datasets. SnappyData has integrated approximate query processing for enabling real-time operational analytics over large (stored or streaming) data. Overall, SnappyData's goal is to yield a significantly lower TCO for mixed workloads compared to using disparate products that are managed, deployed, and monitored separately.

References

Abadi D et al (2013) The design and implementation of modern column-oriented database systems. Found Trends Databases 5(3):197–280

Agarwal S, Panda A, Mozafari B, Iyer AP, Madden S, Stoica I (2012) Blink and it's done: interactive queries on very large data. In: PVLDB

Agarwal S, Mozafari B, Panda A, Milner H, Madden S, Stoica I (2013) BlinkDB: queries with bounded errors and bounded response times on very large data. In: EuroSys

Agarwal S, Milner H, Kleiner A, Talwalkar A, Jordan M, Madden S, Mozafari B, Stoica I (2014) Knowing when you're wrong: building fast and reliable approximate query processing systems. In: SIGMOD

Akidau T et al (2013) MillWheel: fault-tolerant stream processing at internet scale. In: PVLDB

Akidau T et al (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. In: PVLDB

Al-Kateb M, Lee BS (2010) Stratified reservoir sampling over heterogeneous data streams. In: SSDBM

Apache Geode. http://geode.incubator.apache.org/

Apache Samza. http://samza.apache.org/

Barber R, Huras M, Lohman G, Mohan C, Mueller R, Özcan F, Pirahesh H, Raman V, Sidle R, Sidorkin O et al (2016) Wildfire: concurrent blazing data ingest and analytics. In: SIGMOD

Braun L et al (2015) Analytics in motion: high performance event-processing and real-time analytics in the same database. In: SIGMOD

Chandramouli B et al (2014) Trill: a high-performance incremental query processor for diverse analytics. In: PVLDB

Chandrasekaran S et al (2003) TelegraphCQ: continuous dataflow processing. In: SIGMOD

Chaudhuri S, Das G, Narasayya V (2007) Optimized stratified sampling for approximate query processing. ACM Trans Database Syst 32(2):9

CliffGuard. A general framework for robust and efficient database optimization. http://www.cliffguard.org

Exactly-once processing with trident – the fake truth. https://www.alooma.com/blog/trident-exactly-once

Fernandez RC et al (2015) Liquid: unifying nearline and offline big data integration. In: CIDR

Gray J, Lamport L (2006) Consensus on transaction commit. ACM Trans Database Syst 31(1): 133–160

He W, Park Y, Hanafi I, Yatvitskiy J, Mozafari B (2018) Demonstration of VerdictDB, the platform-independent AQP system. In: SIGMOD

Helland P (2007) Life beyond distributed transactions: an apostate's opinion. In: CIDR

Huang J, Mozafari B, Schoenebeck G, Wenisch T (2017) A top-down approach to achieving performance predictability in database systems. In: SIGMOD

IBM. InfoSphere BigInsights. http://tinyurl.com/ouphdss

Indexedrdd for apache spark. https://github.com/amplab/spark-indexedrdd

Liarou E et al (2012) Monetdb/datacell: online analytics in a streaming column-store. In: PVLDB

Makin' Bacon and the Three Main Classes of IoT Analytics. http://tinyurl.com/zlc6den

Meehan J et al (2015) S-store: streaming meets transaction processing. In: PVLDB

Mozafari B (2017) Approximate query engines: commercial challenges and research opportunities. In: SIGMOD

Mozafari B, Zaniolo C (2010) Optimal load shedding with aggregates and mining queries. In: ICDE

Mozafari B, Niu N (2015) A handbook for building an approximate query engine. IEEE Data Eng Bull 38(3):3–29

Mozafari B, Zeng K, Zaniolo C (2012) High-performance complex event processing over xml streams. In: SIGMOD

Mozafari B, Ye Goh EZ, Yoon DY (2015) CliffGuard: a principled framework for finding robust database designs. In: SIGMOD

Mozafari B, Ramnarayan J, Menon S, Mahajan Y, Chakraborty S, Bhanawat H, Bachhav K (2017) SnappyData: a unified cluster for streaming, transactions, and interactive analytics. In: CIDR

Ousterhout K et al (2015) Making sense of performance in data analytics frameworks. In: NSDI

Park Y, Cafarella M, Mozafari B (2016) Visualization-aware sampling for very large databases. In: ICDE

Park Y, Tajik AS, Cafarella M, Mozafari B (2017) Database learning: towards a database that becomes smarter every time. In: SIGMOD

Park Y, Mozafari B, Sorenson J, Wang J (2018) VerdictDB: universalizing approximate query processing. In: SIGMOD

Ramnarayan J, Mozafari B, Menon S, Wale S, Kumar N, Bhanawat H, Chakraborty S, Mahajan Y, Mishra R, Bachhav K (2016) SnappyData: a hybrid transactional analytical store built on spark. In: SIGMOD

SnappyData (2016) Streaming, transactions, and interactive analytics in a unified engine. http://web.eecs.umich.edu/~mozafari/php/data/uploads/snappy.pdf

Thakkar H, Laptev N, Mousavi H, Mozafari B, Russo V, Zaniolo C (2011) SMM: a data stream management system for knowledge discovery. In: ICDE
TIBCO. StreamBase. http://www.streambase.com/
Tian B, Huang J, Mozafari B, Schoenebeck G, Wenisch T (2018) Contention-aware lock scheduling for transactional databases. In: PVLDB
Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, Bhagat N, Mittal S, Ryaboy D (2014) Storm@twitter. In: SIGMOD
Vitter JS (1985) Random sampling with a reservoir. ACM Trans Math Softw 11(1):37–57
Xin R, Rosen J. Project Tungsten: bringing Spark closer to bare metal. http://tinyurl.com/mzw7hew
Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: NSDI
Zamanian E, Binnig C, Salama A (2015) Locality-aware partitioning in parallel database systems. In: SIGMOD
Zeng K, Gao S, Gu J, Mozafari B, Zaniolo C (2014a) ABS: a system for scalable approximate queries with accuracy guarantees. In: SIGMOD
Zeng K, Gao S, Mozafari B, Zaniolo C (2014b) The analytical bootstrap: a new method for fast error estimation in approximate query processing. In: SIGMOD
Zeng K, Agarwal S, Dave A, Armbrust M, Stoica I (2015) G-OLA: generalized on-line aggregation for interactive analysis on big data. In: SIGMOD

Social Networking

▸ Big Data in Social Networks

Spark SQL

Xiao Li, Cheng Lian, and Shu Mo
Databricks Inc., San Francisco, CA, USA

Definitions

SQL is a highly scalable and efficient relational processing engine with ease-to-use APIs and mid-query fault tolerance. It is a core module of Apache Spark, which is a unified engine for distributed data processing (Zaharia et al. 2012). Spark SQL can process, integrate, and analyze the data from diverse data sources (e.g., Hive, Cassandra, Kafka, and Oracle) and file formats (e.g., Parquet, ORC, CSV, and JSON). The common use cases include ad hoc analysis, logical warehouse, query federation, and ETL processing. It also powers the other Spark libraries, including structured streaming for stream processing, MLlib for machine learning (Meng et al. 2016; Michael et al. 2018), GraphFrame for graph-parallel computation (Dave et al. 2016), and TensorFrames for TensorFlow binding. These libraries and Spark SQL can be seamlessly combined in the same application with holistic optimization by Spark SQL.

Overview

Spark is a general purpose big data processing system. It was originally developed in the AMPLab at UC Berkeley and donated to Apache Software Foundation in 2013. Now, Apache Spark is one of the most popular open-source projects in data analytics and query processing. Spark SQL (Armbrust et al. 2015) (its predecessor, Shark (Xin et al. 2013)) was introduced in 2014 and became the core of Apache Spark ecosystem. It enables Spark to perform efficient and fault-tolerant relational query processing with analytics database technologies. The relational queries are compiled to Resilient Distributed Dataset (RDD) transformations and actions, which are executable in Spark.

Spark SQL follows the classic query processing and federation architecture with adoption of the recent research (e.g., whole-stage code generation). Through the user-facing APIs (SQL, DataFrame, and Dataset), the user queries are converted to unresolved abstract syntax trees (called unresolved logical plans). The plans are then analyzed using the session-specific temporary view manager, and the cross-session cache manager and catalog. Logical optimizer and physical planner optimize the resolved plans by applying heuristics-based and cost-based transformation rules. During the query planning phase, the sub-plans are pushed down to the underlying data sources for more

efficient processing, if possible; the logical plans are converted to the executable physical plans consisting of transformations and actions on RDDs with the generated Java code. The code is compiled to Java bytecode, executed at runtime by JVM and optimized by JIT to native machine code at runtime.

Declarative APIs

Prior to Spark SQL, Spark offers low-level procedural APIs for operating RDDs and building DAGs that are executable in Spark. Spark SQL introduces three declarative APIs (DataFrame, Dataset, SQL language), which are complementary to the low-level Spark APIs (i.e., RDD APIs). It facilitates tight integration of relational queries and complex procedural processing.

The SQL API is based on ANSI SQL:2003 standard with full compliance to Hive query language (HiveQL). For the existing Hive users, the compliance facilitates migration to Spark SQL. Spark SQL also provides language-integrated and lazily evaluated DataFrame/-Dataset APIs. The DataFrame API provides untyped relational operations, while the Dataset API provides a typed version for better type safety. The DataFrame API is available in Scala, Java, Python, and R. The Dataset API is only available in Scala and Java since Python and R are dynamically typed languages.

Compared with SQL, DataFrame/Dataset APIs provide richer and user-friendly interfaces, since the APIs are not limited by ANSI SQL compliance and also fully integrated with the programming languages (Java/Scala/Python/R). Using DataFrame/Dataset APIs, a complex dataflow logics can be split to multiple simpler modular host-language functions and then build up a single logical plan for holistic query optimization. All these three APIs are internally represented by the same Catalyst logical plans. They can be mixed, combined, optimized, and executed holistically, thanks to the lazy execution. The execution is triggered until users call the action APIs (e.g., collect, save, and show).

Query Optimization

Spark SQL optimizes the plans using the stratified search strategy that are widely used in the commercial database vendors (e.g., Oracle and DB2). First, the optimizer rewrites the query plans using heuristics-based rules. The typical transformation rules include predicate pushdown, column pruning, outer join elimination, constraint propagation, and predicate inference. Then, the cost-based plan enumeration and method selection are executed. The cost-based optimizer framework is initially shipped with Spark 2.2 and still rapidly evolving. Histogram was introduced for cardinality estimation in Spark 2.3. More accurate cost model, demand-driven enumerators, and adaptive query optimization are in the road map.

Spark SQL optimizer is extensible. Custom optimizer rules can be injected. For example, the users can add extra optimization rules for pushing more operators into the external data source systems or supporting the new data types.

Query Execution

Memory and CPU, rather than disk and network I/O, are the major performance bottlenecks of query execution in Apache Spark (Ousterhout et al. 2017), thanks to the progress in related hardware and data compression. The project Tungsten speeds up query execution by optimizing the efficiency of CPU and memory. The major focuses include off-heap memory management and runtime code generation.

Memory Management

The overhead of JVM objects and GC are significant for data intensive Java applications. Apache Spark leverages the application semantics to explicitly manage the memory using the *sun.misc.Unsafe* feature. This feature provides the C-style direct access to off-heap memory. The memory managed by Spark is invisible to the garbage collector. Tuning

GC for higher performance is not needed. The compiled/interpreted operations directly manipulate the binary data represented in the Tungsten specialized format. Compared with JVM objects, it has less memory footprint and thus reduces the processing time.

Runtime Code Generation

On the driver, Spark generates Java source code specialized to each query. On the executors, the generated code is compiled to Java bytecode by a lightweight Java compiler, which runs directly on the JVM and can be further compiled to native code by the just-in-time (JIT) compiler in the JVM. It strikes a good balance between performance and readability (and thus debuggability) of generated code.

Runtime code generation is performed on two levels: (1) expressions are generated into straightforward Java code to reduce interpretation overhead, and (2) where possible, multiple adjacent physical plan operators, along with the expressions involved, are fused together using a push-based model and generated into a single code generation unit (called a stage). Compared with the original iterator-based pull model (Volcano), this reduces the overhead of virtual function calls and materialization of intermediate results between operators, improves data locality, and enables further specialization with the context of multiple operators.

The same code generation framework is also used to speed up serialization/deserialization. As a unified data processing framework, Spark SQL supports flexible use of UDFs and type-safe Dataset APIs, both of which involve conversions between domain objects and Spark SQL internal data representations.

Data Sources

Spark separates computation and storage. The data sources are autonomous and can be shared with the other processing engines. The data schema is dynamic. The schema is just a virtual view that can be predefined or derived when reading it. For example, the built-in file source connectors, including JSON, CSV, Parquet, and ORC, offer read-time schema inference. All the inferred or predefined schemas can be stored in a global persistent catalog, which is Hive metastore by default. The Hive metastore can also be shared with the other engines (e.g., Hive).

Thanks to the rich interoperability with external data sources, Spark SQL can read from, integrate with, and write to a variety of data sources. Users can use the built-in connectors and also plugin other third-party connectors. Through the data source APIs, third parties can build customized connectors to access the data stores.

Highly efficient vectorized readers are provided for columnar file sources (e.g., Parquet and ORC). Such complicated I/O operations are not fused into the whole-stage codegen. Bulk reading and processing can reduce the per-tuple interpretation overhead and leverage compiler optimization. The built-in cache mechanism is also columnar. The external sources and intermediate results can be explicitly cached in memory or local disk for reuse.

Conclusion

Since the initial release, Apache Spark has quickly become the largest open-source community in big data. It is the work of over 1000 contributors from over 250 companies. Spark SQL is the most active component in Apache Spark. Spark SQL brings end-to-end optimization in the sophisticated applications, which use one or multiple Spark libraries (e.g., SQL, MLlib, and structured streaming). More breakthroughs are expected in the coming years, as Spark SQL is growing to be a compiler of the unified analytics engine.

References

Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, Zaharia M (2015) Spark SQL: relational data processing in spark. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD'15)

Dave A, Jindal A, Li LE, Xin R, Gonzalez J, Zaharia M (2016) Graphframes: an integrated API for mixing graph and relational queries. In: Proceedings of the 4th international workshop on graph data management experiences and systems (GRADES'16)

Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S, Xin D, Xin R, Franklin MJ, Zadeh R, Zaharia M, Talwalkar A (2016) MLlib: machine learning in apache spark. J Mach Learn Res 17(1):34:1–34:7

Michael A, Tathagata D, Joseph T, Burak Y, Shixiong Z, Reynold X, Ali G, Ion S, and Matei Z (2018) Structured Streaming: A declarative API for real-rime applications in apache spark. In: Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD'18). 601–613

Ousterhout K, Canel C, Ratnasamy S, Shenker S (2017) Monotasks: architecting for performance clarity in data analytics frameworks. In: Proceedings of the 26th ACM symposium on operating system principles

Xin RS, Rosen J, Zaharia M, Franklin MJ, Shenker S, Stoica I (2013) Shark: SQL and rich analytics at scale. In: Proceedings of the ACM SIGMOD workshop on the web and databases (SIGMOD'13)

Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX symposium on networked systems design & implementation (NSDI'12)

Spark-Based RDF Query Processors

▸ Framework-Based Scale-Out RDF Systems

SparkBench

John Poelman and Emily May Curtin
IBM, New York, USA

Synonyms

Apache Spark benchmarking; Spark-Bench; CODAIT/spark-bench

Overview

SparkBench is a flexible framework for benchmarking, simulating, comparing, and testing versions of Apache Spark and Spark applications. It provides users three levels of parallelism and a variety of built-in data generators and workloads that allow users to finely tune their setup and get the benchmarking results they need.

Definitions

A framework for benchmarking Apache Spark.

Historical Background

Apache Spark began in 2010 as a research project by Matei Zaharia and others in the Berkeley AMPLab. Following the landmark success of *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing* by Zaharia et al. (2012), Spark continued to gain popularity and usage as its performance gains over traditional MapReduce workflows became evident. Spark continued to grow as well, introducing Python and R APIs, machine learning, graph computation, SQL, and streaming computation.

In 2015, a group of researchers at IBM identified a need in the growing Spark ecosystem for a benchmarking solution that focused specifically on Spark. While benchmarking systems such as the AMPLab Big Data Benchmark and Intel's HiBench, to name just two, focused primarily on comparisons of Spark to other big data compute engines, the researchers at IBM chose to focus more specifically on Spark. In their 2015 paper *SPARKBENCH: A Comprehensive Benchmarking Suite For In Memory Data Analytic Platform Spark*, they wrote,

> While Spark has been evolving rapidly, the community lacks a comprehensive benchmarking suite specifically tailored for Spark. The purpose of such a suite is to help users to understand the tradeoff between different system designs, guide the configuration optimization and cluster provisioning for Spark deployments. Existing big data benchmarks... are either designed for other frameworks such as Hadoop or Hive, or are too general to provide enough insights on Spark workload characteristics. (Li et al. 2015)

SparkBench Versus Other Tools

Intel's HiBench and other similar tools benchmark select Spark algorithms against implementations in other frameworks. These tools focus on a small selection of algorithms that are run in the style of traditional benchmarks. They are not focused specifically on Apache Spark, nor do they claim to be. Instead, they are useful for differentiating Spark against frameworks such as MapReduce, Storm, Hive, and others. SparkBench was designed to exercise various subsystems of Spark in a detailed way, not to compare Spark against other frameworks.

There are a number of projects that provide detail profiling information on Spark performance in a particular job. These, in most cases, can be combined with SparkBench to great effect by using SparkBench to build the simulation or benchmark in question and also drive the profiler. Some projects in this category include CODAIT/spark-tracing (https://github.com/CODAIT/spark-tracing) and the commercially available YourKit (http://spark.apache.org/developer-tools.html#profiling).

Many production users of Spark have also observed the need for a production-grade Spark job scheduling tool. Some examples include Airflow by Airbnb and Azkaban by LinkedIn. While SparkBench provides the ability for users to compose suites of jobs to simulate use cases, it is not a production tool. Airflow and Azkaban provide feature-filled DAG schedulers and production monitoring for Spark jobs. They may be used as drivers for SparkBench to further enable complex simulation of real-world Spark use cases.

SparkBench Version 1.x Structure

The first version of SparkBench included a suite of individually compiled Spark applications. Each contained an application for generating data and an application containing the implementation of the algorithm in question. These applications were largely written in a mix of Java and Python and stitched together through a plethora of bash scripts that provided configuration and coalescing of results. The applications were designed to be individually compiled and run one at a time on a given cluster.

Key to the design of SparkBench 1.0 was the breadth of applications. In 2015, some of the now more widely used components of Spark including Spark SQL and Streaming were only just debuting. The authors of SparkBench 1.0 were careful to include representative workloads for each major piece of Apache Spark.

The source code for the first version of SparkBench was released on Bitbucket. It was eventually ported to GitHub under the umbrella of the CODAIT organization where contributors continued to update the code for subsequent releases of Apache Spark.

Motivation for Rewrite

Since the initial release of SparkBench in 2015, the adoption and number of use cases for Spark have continued to grow at a rapid pace. Notebooks now provide data scientists and analysts new ways of collaborating; cloud services provide businesses new ways of leveraging and accessing the cluster compute power of Spark; and Apache Spark itself has gone from version 1.4.0 to version 2.x in just 2 short years.

In 2017, a team at IBM working on performance improvements to core components of Apache Spark realized the need for a benchmarking tool that could do more than benchmarking. A simple run of traditional algorithms could not capture the complex nature of multiuser user cases that were the focus of the new improvements. To that end, Scala developer Emily May Curtin embarked on a major rewrite of SparkBench.

Project Structure

The new version of SparkBench (2.x) is a complete, ground-up rewrite of version 1.x. While version 1.x was a series of individually compiled Java or Scala programs with their own main() functions, the new version is a single Scala framework compiled using one SBT build file. Rather than algorithm implementations living alone in their own jar stitched together through bash, each is an implementation of the abstract workload class. All configurations are done in a single configuration file rather than a series of scattered variables in various bash scripts.

	Old version	New version
Project design	Series of shell scripts calling individually built jars	Scala project built using SBT
Build/release	Built manually, released manually	SBT, Travis CI, auto-release to GitHub releases on PR merge
Configuration	Scattered in shell script variables	Centralized in one configuration file with many new capabilities
Parallelism	None	Three different levels
Custom workloads	Requires writing a new subproject with all accompanying bash	Bring your own jar, plug it into existing framework with all configuration included

The new version of SparkBench features three independent levels of parallelism to enable users to dial in accurate settings for their particular simulation.

Foundations

SparkBench provides three levels of parallelism in which users can compose workloads and data generators to accomplish their benchmarking goals quickly and easily.

Workloads, Spark configuration, and other experiment structure are defined in one single config file written in HOCON, a superset of JSON. They have a nested structure that starts with spark-submit configs and then goes down into workload suites and finally workloads at the bottom of the nested structure.

Workloads

The atomic unit of organization in SparkBench is the workload. Workloads are stand-alone Spark jobs that read their input data, if any, from disk, and write their output, if the user wants it, out to disk.

Some workloads are designed to exercise a particular algorithm implementation or a particular method. Others are designed to simulate Spark use cases such as multiple notebook users hitting a single Spark cluster.

Some existing categories of workloads include:

- ML workloads: Logistic Regression, K-Means, etc.
- "Exercise" workloads: designed to examine one particular portion of the Spark pipeline. A good example is SparkPi, a compute-heavy workload with very little disk IO.
- Data generators: SparkBench has the capability to generate data according to many different configurable generators. Generated data can be written to any storage addressable by Spark, including local files, HDFS, S3, etc.

Workloads can be composed with one another. They can be launched serially or in parallel.

Workload Suites

Workload suites are logical groups of workloads. Workload suites can be composed with each other for benchmarking tasks or to simulate different cluster use cases.

Workload suites control the benchmark results output and repetition of workloads. They can be composed with each other in serial or parallel fashion. The level of parallelism of the workload suite does not affect the level of parallelism for the workloads contained within each suite.

Spark-Submit Configuration

Under the hood, SparkBench converts users' configuration files into a series of spark-submit scripts. The spark-submit-config section of the configuration file allows users to change the parameters of those spark-submits.

Spark-submits, like other parts of the SparkBench configuration, can be composed serially or in parallel in order to accurately simulate different use cases. Multiple spark-submits additionally allow users to test different versions of Spark, different clusters, different configuration settings on the same cluster, and much more.

As with workloads and workload suites, spark-submits can be composed with each other and can be launched serially or in parallel. These three levels of parallelism (spark-submits, workload suites, and workloads) allow users fine-grained control over the setup of their simulation, so they can dial in the conditions to be as true to life as possible.

Example Configuration File

This is a configuration file used to benchmark performance of SQL queries against the same dataset stored in Parquet and CSV.

The first workload suite generates the data in CSV format, then picks up the CSV it just generated and rewrites it to Parquet format.

The second workload suite runs a total of four different workloads using the cross product of the two parameter lists. These four workloads are repeated ten times.

```
spark-bench = {
  spark-submit-config = [{
    spark-home = "XXXXXXX" // PATH TO
     YOUR SPARK INSTALLATION
    spark-args = {
      master = "XXXXXXX" // FILL IN
        YOUR MASTER HERE
      executor-memory = "XXXXXXX" //
        FILL IN YOUR EXECUTOR MEMORY
    }
    conf = {
      // Any configuration you need
         for your setup goes here,
         like:
      // "spark.dynamicAllocation.
        enabled" = "false"
    }
    suites-parallel = false
    workload-suites = [
      {
        descr = "Generate a dataset,
         then take that same dataset
         and write it out to
         Parquet format"
        benchmark-output = "hdfs:///
        tmp/csv-vs-parquet/results-
        data-gen.csv"
        // We need to generate the
        dataset first through the data
        generator, then we take that
        dataset and convert it to
        Parquet.
        parallel = false
        workloads = [
          {
            name = "data-generation-
                   kmeans"
            rows = 10000000
            cols = 24
            output = "hdfs:///tmp/csv-
                   vs-parquet/kmeans-
                   data.csv"
          },
          {
            name = "sql"
            query = "select * from
                   input"
            input = "hdfs:///tmp/csv-
                   vs-parquet/kmeans-
                   data.csv"
            output = "hdfs:///tmp/csv-
                   vs-parquet/kmeans-
                   data.parquet"
          }
        ]
      },
      {
        descr = "Run two different SQL
        queries over the dataset in
        two different formats"
        benchmark-output = "hdfs:///
        tmp/csv-vs-parquet/results-
        sql.csv"
        parallel = false
        repeat = 10
        workloads = [
          {
            name = "sql"
            input = ["hdfs:///tmp/csv-
            vs-parquet/kmeans-data.
            csv", "hdfs:///tmp/csv-vs-
            parquet/kmeans-data.
            parquet"]
            query = ["select * from
            input", "select `0`, `22`
            from input where `0` <
            -0.9"]
            cache = false
          }
        ]
      }
    ]
  }]
}
```

Key Applications

SparkBench is in active use by developers working on contributions to Spark core. These developers use the project to compare the runtimes of

standard algorithms in the latest official version of Spark against their branch with changes.

Other developers have used the custom workload plugin capability of SparkBench to create custom workloads that exercise a very specific subsystem of Spark such as the caching mechanisms in core as they relate to dynamic allocation.

SparkBench additionally provides mechanisms for developers to simulate notebook environments such as Jupyter or Apache Zeppelin. By inserting a Sleep workload between workloads, SparkBench users can build up a simulation of a notebook user. It is a simple extension to copy this workload suite to simulate multiple users.

This notebook simulation technique has been used to great effect by developers in the IBM Spark Technology Center who used SparkBench to debug and benchmark changes to Spark core involving resource contention settings.

Future Work

SparkBench is under active development by IBM and contributors from all over the world. Planned future work centers around several efforts.

1. *Launching through interfaces other than spark-submit (Livy, other REST services).* The structure of SparkBench should reflect the myriad of ways that users can interface with Spark. Adding REST capability to SparkBench allows for a new variety of use cases to be covered such as Spark as a service setups and other remote architectures.
2. *Expanding selection of built-in workloads.* At the time of publication, IBM developers are working on porting many workloads from version 1.x to 2.x.
3. *Allowing tighter integration with pluggable listeners and profilers.* SparkBench sits at the level of the application. To get more detailed profiling data on specific functions and do system profiling, users can employ a variety of profilers and tracers. In the future the output of SparkBench should be synced and packaged with the output of tracers.
4. *Increase the number of standard big data benchmarks available by default, particularly TPC-DS.* The TPC-DS dataset and associated 100 SQL queries are a key standard benchmark for SQL systems. At the time of publication, developers at IBM are in progress adding workloads for TPC-DS data generation as well as the standard queries.

Cross-References

- Apache Spark
- SparkBench
- TPC-DS

References

AMPLab Big Data Benchmark. https://amplab.cs.berkeley.edu/benchmark. Accessed 23 Feb 2018

Apache Airflow. http://airbnb.io/projects/airflow/. Accessed 23 Feb 2018

Apache Spark. https://spark.apache.org/. Accessed 23 Feb 2018

Apache Zeppelin. https://zeppelin.apache.org/. Accessed 23 Feb 2018

Azkaban. https://azkaban.github.io/. Accessed 23 Feb 2018

HOCON (Human-Optimized Config Object Notation). https://github.com/lightbend/config/blob/master/HOCON.md. Accessed 23 Feb 2018

IBM Spark-Tacing. https://github.com/CODAI/spark-tracing. Accessed 23 Feb 2018

Intel HiBench Suite. https://github.com/intel-hadoop/HiBench. Accessed 23 Feb 2018

Li M et al (2015) SparkBench: a comprehensive benchmarking suite for in memory data analytic platform Spark. https://research.spec.org/fileadmin/user_upload/documents/wg_bd/BD-20150401-spark_benchmark-v1.3-spec.pdf. Accessed 23 Feb 2018

Project Jupyter. http://jupyter.org/. Accessed 23 Feb 2018

TPC Decision Support Benchmark. http://www.tpc.org/tpcds/default.asp. Accessed 23 Feb 2018

YourKit Java Profiler. https://www.yourkit.com/java/profiler/features/. Accessed 23 Feb 2018

Zaharia M et al (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. http://people.csail.mit.edu/matei/papers/2012/nsdi_spark.pdf. Accessed 23 Feb 2018

Spark-Bench

▸ SparkBench

SPARQL Federation

▸ Federated RDF Query Processing

Spatial and Textual Data

▸ Spatio-textual Data

Spatial Data

▸ Spatio-social Data
▸ Spatial Data Mining

Spatial Data Integration

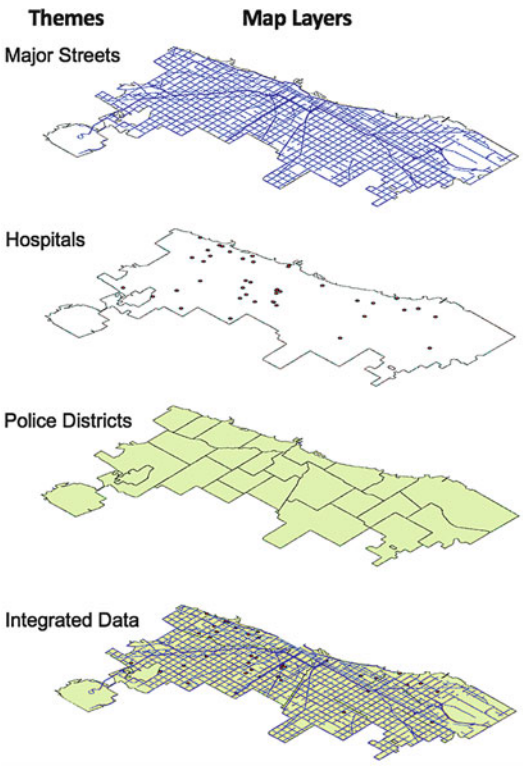
Booma Sowkarthiga Balasubramani and Isabel F. Cruz
University of Illinois at Chicago, Chicago, IL, USA

Synonyms

Geospatial data integration; Spatiotemporal Data Integration

Definitions

Spatial data integration is a process in which different geospatial datasets, which may or may not have different spatial coverages, are made compatible with one another (Flowerdew 1991). The goal of spatial data integration is to facilitate the analysis, reasoning, querying, or visualization of the integrated spatial data. Figure 1 illustrates the integration of three layers or *themes*: major streets, hospitals, and police districts of the City of Chicago (Chi 2017).



Spatial Data Integration, Fig. 1 Spatial data integration (Chi 2017)

Spatial data (often used synonymously with *geospatial data* and *geographical data*) refers to the information – location, shape, and relationships among geographic features – that describes the distribution of things on the surface of the Earth (DeMers 2008; Walker 1993).

Geospatial datasets that relate to physical geography, such as land and ocean boundaries, topography, weather, hydrology, natural disasters, land cover, and to human geography, such as administrative boundaries, land use, population, crime, buildings, water pipes, roads, points of interest, and many more, are freely available from a wide variety of sources (e.g., Socrata, data.gov). Many of those datasets vary both with respect to time and space. While each dataset may be important in its own right, it is often

their joint analysis that makes them critical for decision-making. Examples include finding the lakes in Maine (Egenhofer 2002), returning the towns affected by fires (Kyzirakos et al. 2014), or estimating the number of people affected by a water main break (Balasubramani et al. 2017).

Overview

Data are generated by various sources and different stakeholders, and they correspond to different locations and times. This section discusses such characteristics of spatial data that influence the techniques for spatial data integration.

Spatial and Spatiotemporal Data Types

Spatial data can be associated with (Flowerdew 1991):

Points: Sample points, selected randomly or for convenience (points in a survey, rain gauges), or points representing real objects (trees, buildings, cities).

Lines: Sample lines (paths for ecological sampling) or real phenomena (streets, rivers, geological faults).

Areas: With natural boundaries (islands, marshes) or artificial boundaries (land parcel, census tract).

Surfaces: A phenomenon defined across a region but measured only at discrete points (rainfall). Values at other points can be estimated (e.g., by interpolation).

For spatiotemporal data, we consider a single data type (Zheng 2015; Yoon and Shahabi 2008):

Trajectories: A trajectory of a moving object is a sequence of spatial points sampled at discrete instances of time. It is represented by a series of chronological points, $(p_1, t_1), (p_2, t_2), \ldots, (p_n, t_n)$, where p_i is a two- or three-dimensional vector corresponding to the geospatial position observed at a time stamp t_i, where $i = 1, \ldots, n$.

Spatial Data Formats

Spatial data are represented using one of the two data formats (Maffini 1987):

Vector: A coordinate-based data format that represents spatial data in the form of points, lines, and polygons. This format allows for the efficient encoding and operations to determine, for example, the eight basic topological relationships (Egenhofer 1993), namely, *disjoint*, *meet*, *overlap*, *contains*, *inside*, *covers*, *covered by*, and *equal*.

Raster: A data format that is composed of cells distributed in a regular grid, each containing an attribute value and location coordinates (Bishr 1998). This format encodes continuous data efficiently, such as in a topographic map that represents elevation using a color scale.

Conflation is the process of combining two or more sources representing the same geographic information so as to retain accurate data, minimize redundancy, and reconcile data conflicts (Longley 2005). The techniques for conflation vary based on the spatial data formats (Saalfeld 1988; Cobb et al. 1998). For example, vector-to-vector conflation involves matching vector data based on the similarities of the shapes of the vectors, geometric information, or the vector attributes. Vector-to-raster conflation exploits auxiliary spatial information (e.g., coordinates from a satellite image) and nonspatial information (e.g., the image resolution or color) and performs image processing to determine the correspondence among the datasets. Raster-to-raster conflation involves data-specific image processing techniques to extract and match various features (e.g., edges) across images (Dare and Dowman 2000).

Spatial Transformations

Spatial transformations are functions that obtain new spatial data from existing data, for example (Longley et al. 2015):

Buffering: Produces a new geometry by adding an area around the original objects (points, lines, areas).

Point-in-Polygon: Determines whether a given point lies inside or outside a given polygon.

Polygon Overlay: Outputs the polygons resulting from the intersections of two polygons.

Spatial Interpolation: Uses information from sampled points to guess the values at other points.

Density Estimation: Constructs an estimate of an unknown probability density function, based on observed data.

Heterogeneity

Differences in spatial data types and formats lead to two forms of heterogeneity that need to be resolved when integrating spatial data. Other obstacles for integrating data stem from differences in syntax, schema, or semantics (Bishr 1998). In addition, different data models, update frequency, resolution, and scale are also to be considered.

Spatial data exhibit various kinds of heterogeneity (Beck et al. 2008; Hakimpour 2003; Bishr 1998):

Syntactic Heterogeneity: Spatial data is stored in different formats (e.g., shapefile, KML). The data format dictates the tools and approaches to query, integrate, and analyze the data.

Structural or Schematic Heterogeneity: This kind of heterogeneity arises when the data model is the same (e.g., relational) but the schemas differ, including the type of an attribute (for example, ºF vs. ºC). Further heterogeneities occur when the same concept is in one of the table names in a schema and is a name of an attribute in another schema.

Semantic Heterogeneity: Semantic heterogeneities occur when the same entity has different names, or when different entities have the same name. For example, *Chicago* and *Windy City* represent the same entity, whereas *Square* represents a geometry with four sides of equal length or a *City Square*.

Spatial Modeling Heterogeneity: The same geographical phenomena may be modeled differently, for example, *street* as a *line* in one dataset, and as a *polygon* in another dataset.

Resolution or Granularity: There are three kinds of resolution: *spatial*, *temporal*, or *semantic*. *Spatial resolution* is the grain size or the cell size of spatial data, and represents the amount of detail that can be observed in the spatial dataset (Longley 2005). For instance, satellite imagery with a resolution of 2.4 m is more detailed than a satellite imagery with a resolution of 1.1 km. *Temporal resolution* is the time interval during which the value remains the same, for example, temperature that is published every hour, but the estimated time of arrival for a flight may be updated every five minutes. *Semantic resolution* refers to the level of specification of a real-world entity (Fonseca et al. 2002). For example, a body of water may be perceived as a lake or as a water body, depending on the application. Semantic similarity can be measured by the distance that separates two concepts in a classification hierarchy (e.g., an ontology).

Scale: Often used synonymously with resolution, scale describes the dimensions of spatial phenomena and determines its object type (Longley 2005). The details of the spatial phenomena represented in the spatial dataset heavily depends on scale. For example, the New York city is represented as a one-dimensional point on the world map, but is represented as a two-dimensional area on the city map, which is more detailed.

Key Research Findings

Designing spatial data integration systems is characterized by various challenges that spatial data exhibit. Quite justifiably, significant research effort has been invested over the past decades in the spatial data integration problem. This section discusses some of the most important works that focus on integrating spatial data in a seamless manner.

Spatial Entity Resolution

Entity resolution in spatial data integration refers to the process that determines whether two locations referenced by two or more different data

sources correspond to the same real-world entity (Sehgal et al. 2006). Some common approaches for entity resolution in spatial data integration are as follows:

One-Sided Nearest-Neighbor Join: This method associates data with location *a* in dataset *A* to data with location *b* in dataset *B*, if *b* is the closest location to *a* among all other locations in *B* (Batini and Scannapieco 2016).

Image Processing Techniques: Integration of digital maps (Doytsher et al. 2001; Saalfeld 1988) involves image processing techniques and requires complete information about the spatial entity. Satellite imagery has long been used to integrate maps.

Feature-Based Matching: These algorithms use location features, that is, attributes such as spatial coordinates, location name, and location type independently, and find mappings based on a similarity metric such as *edit distance* or *coordinate similarity*.

Feature-Based Integration: This method combines different similarity measures. For example, a feature vector consisting of the similarity metrics for spatial and nonspatial features (e.g., *edit distance* and *coordinate similarity*) is associated with each pair of locations. Final similarity is determined using is performed using machine learning techniques like logistic regression or neural networks.

Uncertainty

Uncertainty, which is the difference between an occurrence in reality and its representation in a database (Zhang et al. 2014), is omnipresent in geospatial data. Uncertainty occurs in several phases ranging from geographical abstraction, data acquisition, limited observations, different but equally valid interpretations of the geography, geo-processing techniques, and representation of the results of geographic computations to the use of the results (Devillers and Jeansoulin 2006; Zhang and Goodchild 2002). When datasets are integrated, the resulting uncertainty stems from the individual datasets as well as from the integration process itself.

Spatial Data Integration: Observations

This section highlights some details observed during spatial data integration (Goodchild et al. 1992; Cruz and Xiao 2008; Mohammadi et al. 2006; Cruz et al. 2007):

1. The set of functions that can be applied or the type of analysis that can be performed on spatial data heavily depends on the data models and probability distribution functions. For example, interpolation can be performed on point samples of mineral deposits to determine the distribution of mineral reserves for an area. However, one cannot interpolate the point samples representing occupancy in high-rise buildings to estimate the population of a city.
2. Data comes from various sources such as social media, private and public organizations, and government agencies. Each of these sources handles data using different database management techniques, formats, and scales. Even in the same organization, for example, in a city, the boundaries of the various districts (school, asphalting, sanitation, forestry) may not coincide, thus complicating correlations across different datasets.
3. The same data can be heterogeneously represented depending on the data provider, either because of the data collection method, storage and management techniques, or, sometimes, the functions used to derive the data.
4. The availability of metadata does not guarantee in itself that data will be correctly integrated. The semantics of the different datasets, the target model in the case of cross-domain data integration, the goal of the integration, and other factors are important.
5. Attribute-level metadata, which contains the data accuracy, date collected/updated, or source of a particular attribute, is usually more useful than the generic theme-based metadata, which focuses on the domain of the spatial data (e.g., agriculture, environmental) and consists of information including the spatial extent, content type, and publisher.
6. Sources with different privacy requirements further add to the data heterogeneity and to

the complexity of the data integration process. For example, data may have to be previously transformed to a coarser detail or integrated using privacy-preserving techniques. Another known risk stems from the fact that the integrated datasets may allow for a more detailed analysis than was previously possible for each dataset.

State-of-the-Art Techniques for Spatial Data Integration

Due to the complex nature of spatial data, a data integration framework cannot be generalized to all spatial data integration cases. Geostatistical techniques are specific to spatial data (Gotway and Young 2002), while others that use ontologies can be used for the integration of spatial (Fonseca et al. 2002) and nonspatial datasets (Cruz and Xiao 2005). Ontologies are knowledge representation structures that define concepts and their relationships explicitly.

Geostatistical Techniques

Geostastical techniques for data integration aim to allow for the synthesis of data at different scales. One of the problems is the fact that spatial data is organized, subdivided, and stored based on spatial units (e.g., land parcel, grid cell of size 1 km^2), which may differ across datasets.

Spatial data transformations are special cases of the *change of support problem (COSP)*, where *support* includes the geometrical size and shape of a region associated with a measurement (Gotway and Young 2002). Examples of COSPs include block kriging and areal interpolation. For example, block kriging applies to the case where point values are observed but the nature of the process is associated with an area, whereas areal interpolation applies to the case where area values are observed for a set of source polygons (e.g., school districts) but need to be converted to a set of target polygons (e.g., census tracts).

Ontology-Based Approaches

To integrate datasets that exhibit different spatial (or temporal) resolutions and are different in terms of schema (structure that represents the organization of data) and semantics, an ontology can act as a mediator between the two datasets so as to establish relationships among the attributes of the datasets (Tran et al. 2015). For example, an urban spatial ontology can be used to establish relationships such as Chicago *is inside* Illinois. It is also possible to use an ontology for each dataset and establish correspondences between the ontologies, using rules defined by the domain experts (Mena et al. 2000). Also, there are different types of ontologies with respect to their expressiveness.

A set of mappings, which defines the correspondences among the concepts in the ontologies, are established to describe topological relations (such as *disjoint*, *meet*, or *overlap*). For resources with rich quantitative spatial information, the focus is on obtaining mappings between spatially colocated regions. The similarity between two geometric shapes is then obtained using the Hausdorff distance, which is defined as the maximum distance of a set to the nearest point in the other set (Rote 1991).

However, two concepts may be matched even if there are other pairs that are better matched. Structural methods (Cruz and Sunna 2008; Melnik et al. 2002) consider the structure around each of the two concepts being matched, that is, the concepts that are adjacent to them. Then, the two concepts match if their adjacent concepts also match.

Examples of Application

Apart from facilitating complex geospatial data analytics, spatial data integration also enables data accessibility (i.e., data storage and retrieval) and data interoperability (i.e., compatibility with other datasets). Some applications that benefit from spatial data integration are described next:

Utility Data Integration: This application facilitates the exploration and analysis of the impact of disruptions in urban infrastructure systems. There are several frameworks that address the challenges in integrating utility data from different perspectives (Beck et al. 2007, 2008; Psyllidis et al. 2015).

Land-use Matching: This application maps two different land-use ontologies (Cruz and Xiao 2008; Cruz et al. 2007), which classify land parcels based on their usage, for example, residential, industrial, agricultural, or commercial, so that a single query can retrieve those parcels across different classifications.

Spatial Phenomena Correlation: This application computes the correlations between different spatial phenomena such as crop rotations and biodiversity (Tran et al. 2015) by integrating a land-use dataset and a biology dataset.

Environmental and Disaster Analysis: This application uses the query language SPARQL extended with geometric objects, set operations on those objects (e.g., union), and the result of spatial transformations on those objects (e.g., buffer), to ask queries that return the names of the towns that have been affected by fire and to construct new geometric objects, for example, the burnt areas in coniferous forests (Kyzirakos et al. 2014).

Future Directions for Research

Each of the challenges associated with big data – volume, variety, variability, velocity, and value – is patent in geospatial data integration. A comprehensive environment for integrating and analyzing geospatial data has several layers, each being conceptually complex when considered on its own and even more so when considering the interactions among them. Those layers include data extraction, data translation, ontology extraction, syntactic and semantic matching, spatiotemporal matching, application, visualization, and analytics, together with two crosscutting layers: machine learning and querying (Cruz et al. 2013).

Ontology-based approaches resolve structural and semantic heterogeneities. These are expressive techniques that enable cross-database queries and reasoning on the spatial, temporal, as well as the spatiotemporal relationships. However, ontologies are often not available for the domain of interest and even when available can be location dependent. For example, an ontology of crime in Chicago may not apply to New York City. Land-use ontologies also vary across municipalities and counties (Wiegand et al. 2002) but can be created from the corresponding taxonomies. It is, however, nontrivial to incorporate the associated unstructured information in the ontology. Hence, the creation, reuse, and evolution of ontologies need to be streamlined.

In addition to the expressiveness issues, other issues pertain to the computational efficiency, for example, of record linkage (Fellegi and Sunter 1969; Kelley 1984), large ontology matching (Faria et al. 2017), or geostatistical scale mapping (Gotway and Young 2002). When several of these strategies need to be applied, their optimization is necessary, possibly along with the determination of the order of execution of the various layers to improve overall efficiency.

Cross-References

- Data Integration
- Holistic Schema Matching
- Probabilistic Data Integration
- Record Linkage
- Schema Mapping
- Uncertain Schema Matching

References

(2017) Open data portal of the city of Chicago. https://data.cityofchicago.org/. Accessed: 30 Nov 2017

Balasubramani BS, Belingheri O, Boria ES, Cruz IF, Derrible S, Siciliano MD (2017) GUIDES–geospatial Urban infrastructure data engineering solutions. In: ACM SIGSPATIAL international conference on advances in geographic information systems

Batini C, Scannapieco M (2016) Data and information quality: dimensions, principles and techniques. Springer, Cham

Beck AR, Fu G, Cohn AG, Bennett B, Stell JG (2007) A framework for utility data integration in the UK. In: 26th urban data management symposium, Stuttgart, 10–12 Oct 2007. Taylor & Francis, pp 261–276

Beck AR, Cohn AG, Sanderson M, Ramage S, Tagg C, Fu G, Bennett B, Stell JG (2008) UK utility data integration: overcoming schematic heterogeneity. In: International conference on advanced optical materials

and devices. International Society for Optics and Photonics, 71431Z
Bishr YA (1998) Overcoming the semantic and other barriers to GIS interoperability. Int J Geogr Inf Sci 12(4):229–314
Cobb MA, Chung MJ, Foley III H, Petry FE, Shaw KB, Miller HV (1998) A rule-based approach for the conflation of attributed vector data. GeoInformatica 2(1):7–35
Cruz IF, Sunna W (2008) Structural alignment methods with applications to Geospatial ontologies. Trans GIS: Special Issue on Semant Similarity Meas Geospat Appl 12(6):683–711
Cruz IF, Xiao H (2005) The role of ontologies in data integration. J Eng Intell Syst 13(4): 245–252
Cruz IF, Xiao H (2008) Data integration for querying geospatial sources. Springer, Boston, pp 113–137
Cruz IF, Sunna W, Makar N, Bathala S (2007) A visual tool for ontology alignment to enable geospatial interoperability. J Vis Lang Comput 18(3):230–254
Cruz IF, Ganesh VR, Caletti C, Reddy P (2013) GIVA: a semantic framework for geospatial and temporal data integration, visualization, and analytics. In: ACM SIGSPATIAL international symposium on advances in geographic information systems (ACM GIS). ACM, pp 544–547
Dare P, Dowman I (2000) A new approach to automatic feature based registration of SAR and SPOT images. Int Arch Photogramm Remote Sens 33(B2):125–130
DeMers MN (2008) Fundamentals of geographic information systems. Wiley, Hoboken
Devillers R, Jeansoulin R (2006) Fundamentals of spatial data quality. ISTE Publishing Company, London/Newport Beach
Doytsher Y, Filin S, Ezra E (2001) Transformation of datasets in a linear-based map conflation framework. Surv Land Inf Syst 61(3):165–176
Egenhofer M (1993) A model for detailed binary topological relationships. Geomatica 47(3):261–273
Egenhofer MJ (2002) Toward the semantic geospatial Web. In: ACM symposium on advances in geographic information systems. ACM GIS, pp 1–4
Faria D, Pesquita C, Mott I, Martins C, Couto FM, Cruz IF (2017, in press) Tackling the challenges of matching biomedical ontologies. J Biomed Semant 9:1–19
Fellegi IP, Sunter AB (1969) A theory for record linkage. J Am Stat Assoc 64(328):1183–1210
Flowerdew R (1991) Spatial data integration. Geogr Inf Syst 1:375–387
Fonseca F, Egenhofer M, Davis C, Câmara G (2002) Semantic granularity in ontology-driven geographic information systems. Ann Math Artifi Intell 36(1–2):121–151
Goodchild M, Haining R, Wise S (1992) Integrating GIS and spatial data analysis: problems and possibilities. Int J Geogr Inf Syst 6(5):407–423
Gotway CA, Young LJ (2002) Combining incompatible spatial data. J Am Stat Assoc 97(458):632–648
Hakimpour F (2003) Using ontologies to resolve semantic heterogeneity for integrating spatial database schemata. Ph.D. thesis, Universität Zürich
Kelley RP (1984) Blocking considerations for record linkage under conditions of uncertainty. Technical Report CENSUS/SRD/RR-84/19, U.S. Bureau of the Census, Statistical Research Division, Washington, DC, 709133
Kyzirakos K, Karpathiotakis M, Garbis G, Nikolaou C, Bereta K, Papoutsis I, Herekakis T, Michail D, Koubarakis M, Kontoes C (2014) Wildfire monitoring using satellite images, ontologies and linked geospatial data. J Web Sem 24:18–26
Longley P (2005) Geographic information systems and science. Wiley, Hoboken, pp 148–149
Longley PA, Goodchild MF, Maguire DJ, Rhind DW (2015) Spatial data analysis, chap 13, 4th edn. Wiley, Hoboken
Maffini G (1987) Raster versus vector data encoding and handling: a commentary. Photogramm Eng Remote Sens 53(10):1397–1398
Melnik S, Garcia-Molina H, Rahm E (2002) Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In: IEEE international conference on data engineering (ICDE), pp 117–128
Mena E, Illarramendi A, Kashyap V, Sheth AP (2000) OBSERVER: an approach for query processing in global information systems based on interoperation across pre-existing ontologies. Distrib Parallel Databases 8(2):223–271
Mohammadi H, Binns A, Rajabifard A, Williamson IP et al (2006) Spatial data integration. In: 17th UNRCC–AP conference and 12th meeting of the PCGIAP, Bangkok, pp 18–22
Psyllidis A, Bozzon A, Bocconi S, Bolivar CT (2015) A platform for Urban analytics and semantic data integration in city planning. In: International conference on computer–aided architectural design futures. Springer, pp 21–36
Rote G (1991) Computing the minimum hausdorff distance between two point sets on a line under translation. Inf Process Lett 38(3):123–127
Saalfeld A (1988) Conflation: automated map compilation. Int J Geogr Inf Syst 2(3):217–228
Sehgal V, Getoor L, Viechnicki PD (2006) Entity resolution in geospatial data integration. In: ACM international symposium on advances in geographic information systems. ACM GIS, pp 83–90
Tran BH, Plumejeaud-Perreau C, Bouju A, Bretagnolle V (2015) A semantic mediator for handling heterogeneity of spatio-temporal environment data. In: Metadata and semantics research. Springer, Cham, pp 381–392
Walker R (1993) AGI standards committee GIS dictionary. Association for Geographic Information, London
Wiegand N, Patterson D, Zhou N, Ventura S, Cruz IF (2002) Querying heterogeneous land use data: problems and potential. In: National conference for digital government research (dg.o), pp 115–121
Yoon H, Shahabi C (2008) Robust time-referenced segmentation of moving object trajectories. In: IEEE

international conference on data mining (ICDM). IEEE, pp 1121–1126
Zhang J, Goodchild MF (2002) Uncertainty in geographical information. CRC Press, London
Zhang J, Atkinson P, Goodchild MF (2014) Scale in spatial information and analysis. CRC Press, Boca Raton
Zheng Y (2015) Trajectory data mining: an overview. ACM Trans Intell Syst Technol (TIST) 6(3):29

Spatial Data Mining

Shuliang Wang and Tisinee Surapunt
Beijing Institute of Technology, Beijing, China

Synonyms

SDM theory; Spatial data

Definitions

The growth of spatial data which plays a part in the agricultural products, sustainable development, and human society development is accumulated continuously. Not only the size and volume are immense, the structure is also convoluted with the abundant and deep of their contents. The spatial dataset is full of the information and experience collection from geomatics that relates to Remote Sensing (RS), Global Positioning System (GPS) and Geographic Information System (GIS). A wide variety of databases consist of electronic maps and planning network from their infrastructure. With the increase in the spatial data collection, the processes of gathering, management, and transmission data require the powerful techniques. The traditional methods lag of the ability of big data query. Thus, the Spatial Data Mining (SDM) becomes the suitable technique. The Knowledge Discovery from Geographical Information System database (KDG) approach can support SDM to be more developed in data mining and geomatics (Li and Cheng 1994). The discovered knowledge from spatial datasets can support a decision-making system on various areas such as urban planning and construction, transportation, resource allocation, capital optimization, marketing, and medical treatment. Then, SDM benefits the global sustainability. However, the large volume of spatial datasets will impact definitely the execution time. Then, the computerization which is composed of a computer chip, a power of a central processing unit, and a transfer rate of communication channel will boost up an acceleration of an interactive process. Presently, the artificial intelligence (AI) and machine learning (ML) are important roles instead of the manual process. AI is in charge with the human deterministic intelligence simulation by three strategies which are symbolism, connectionism, and behaviorism. While the ML supervises on the human learning simulation which obtains the knowledge form expert systems automatically, SDM will request the AI and ML to improve the data mining capability.

Recently SDM is popular to present the discovered knowledge through the real-world application but should concern over the crisp data or uncertain data. In that case, the probability theory, spatial statistics, rule induction, clustering analysis, spatial analysis, fuzzy set, data fields, rough sets, genetic algorithms, visualization decision trees can support those issues (Ester et al. 2000).

Overview

SDM Concepts

The SDM can enhance the human ability for extracting, applying, and transforming the spatial data in the real-world application. Because the ML and AI belong to the SDM procedure, the SDM is more understandable by both mathematic and non-mathematic theories. So, the SDM is a technique to clean, sample, and convert a vast and complex spatial data.

The SDM Pyramid is a symbol of SDM Concepts which concentrates on transforming from the spatial data to information and knowledge. The process starts with the data preparation, data mining, and post-processed of data mining. If the description is more abstract,

coherent, and general, the technologies will be more deep and advanced. Piatetsky-Shapiro (1994) reviewed the concept of data, information, and knowledge pyramid (DIKP). The study proposed that the different levels of data presented the different concept element. So, it was impossible to make the association between each concept. Han et al. (2012) demonstrated the visual process of data mining by different graphics. However, their work was imperfect to present the role and difference of elements. It implied that the complexity of spatial data was difficult to depict the levels of data. It is compulsory to integrate DIKP and the process of SDM in order to clarify the concept and roles of SDM. Thus, the SDM Pyramid in Fig. 1 is presented as a result. The SDM Pyramid specifies the level of spatial data from bottom to top. Each level of data presents the different element concept. The SDM pyramid is the main solution of transferring and managing data between layers. For example, the RS images, the images data are analysed and usually take the resolution in distinct layers. The top layer is the coarsest resolution and small scale, while the bottom layer is the finest resolution and largest scale. Then, the data is indexed and well-organized can browse in the multi-source, multi-scale, and cross-resolution image. The SDM pyramid manifests the data characteristics from spatial knowledge to the spatial data in real world. The amount of spatial data directs variation to the data complexity. The more simple data is, the more spatial data quantity contains.

Spatial Data to Spatial Knowledge

The SDM applies an interdisciplinary methodology to acquire spatial knowledge effectively. Each unit of spatial numerical values, spatial data, spatial information, and spatial knowledge presents the dissimilar concepts but connects in detail.

- Spatial Numerical

The spatial numerical represents between a specific value and a measurement unit. The spatial numerical is a transmission when the spatial objects are gathered, transformed, and applied by the computerized SDM.

- Spatial Data

The spatial data can represent raw data or processed data. The processed data illustrates digital and nondigital data such as numbers, words, symbols, graphics, images, video, and language which utilizes the raw data. The spatial data can be a reference in order to know spatial objects and numerical nature with various attributes.

- Spatial Concept

The spatial objects together with their implication and extension describe the spatial concept. The spatial concept may involve in the problem-solving and indicate the states of spatial objects

- Spatial Information

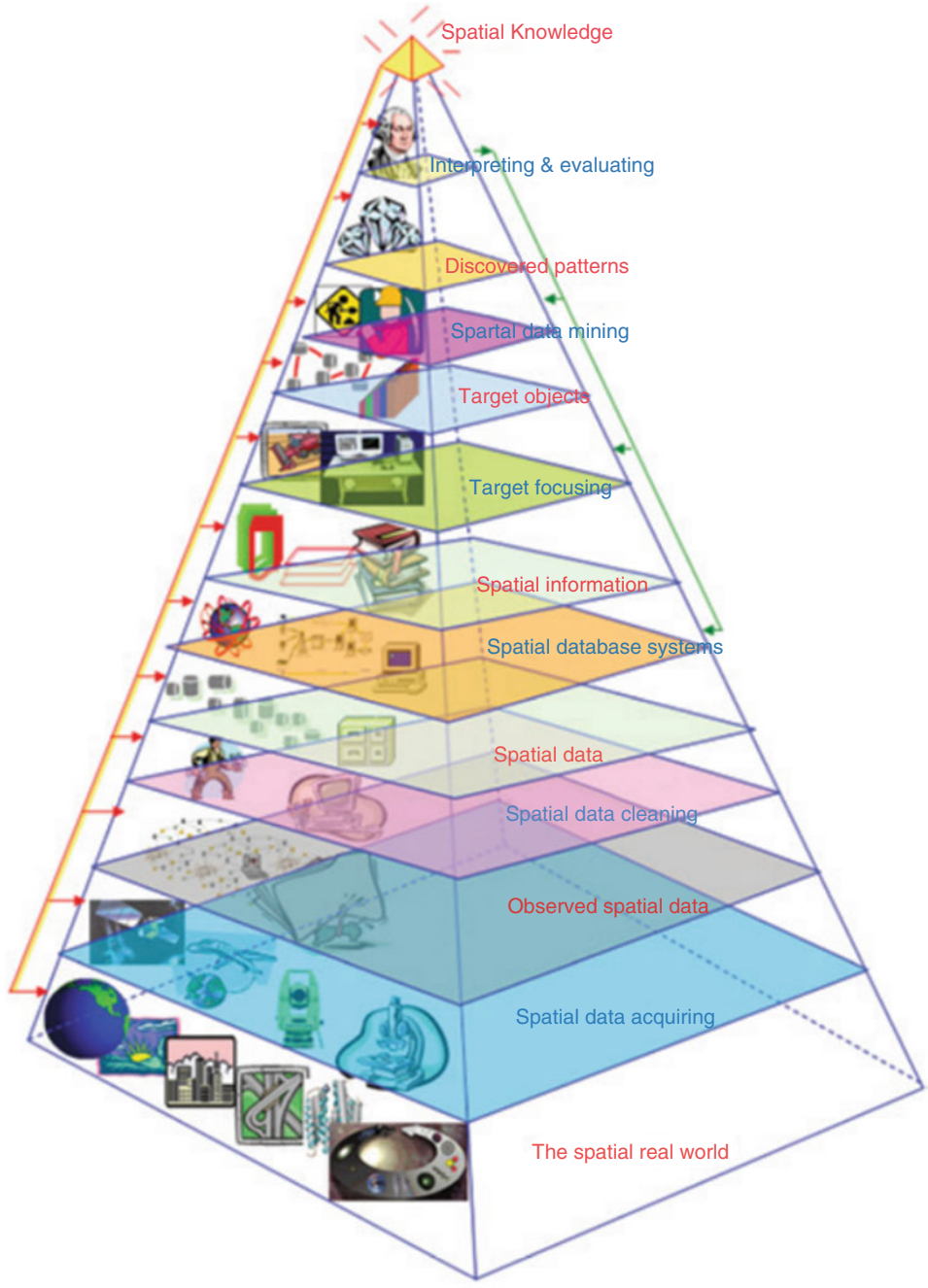
The analysis of spatial objects in the dataset can determine the spatial information. The spatial information demonstrates the meaningfulness and eliminates the possible uncertainty. It is imperative to be the decision-maker.

- Spatial Knowledge

The spatial knowledge is built from one or more parts of the spatial information. The spatial knowledge is depicted through the processes of correlation, association, classification, and clustering. Both spatial information and spatial knowledge require SDM in the data processing.

- Unified Action

The spatial data carries information to specify their properties, quantities, location, and relationship. So, the unified actions of spatial numerical values, spatial data, spatial concept, and spatial information become the spatial knowledge.



Spatial Data Mining, Fig. 1 SDM Pyramid (Li et al. 2015)

Spatial Knowledge to Discover

In an effort to discover the various spatial knowledge, SDM requires various techniques under "rule plus exception" with different views. The miscellaneous preset rules have already qualified on the practical manipulation such as General Geometric Rule and Spatial Association Rule, Spatial Characteristics Rule and Discriminate Rule,

Spatial Clustering Rule, Classification Rule, Spatial Predictable Rule and Serial Rule and Spatial Exception or Outlier. Therefore, the spatial knowledge is extracted properly to be the Discover (Li and Du 2007).

Spatial Knowledge Representation

SDM promotes the spatial knowledge representation in order to demonstrate the knowledge. The integration of traditional expression methods can enhance the conversion of the knowledge expression method efficiently. When the knowledge extraction finishes, the knowledge measure can bound the level to depict the discovered pattern which is meaningful and interesting. A natural language, predicates logic, function model, characteristic table, semantic network, and Petri net are fundamental representations. For example, the natural language which is a general knowledge recognition of uncertainty can interpret spatial datasets based on human thinking. It is an effective tool to reorganize the thinking process. The concept is to map the object from the objective to the subjective cognition. Hence, the more the knowledge is abstract, the more the natural language is proper.

SDM Data Source

Contents and Characteristics of Spatial Data

The primitive data sources of spatial data are geographic contents. The spatial data acquisition gathers from radar, infrared, photoelectric, satellite, digital cameras and telescopes. There are three acquisition methodologies which are point acquisition, area acquisition, and mobility acquisition. The point acquisition is performed by the GPS receivers which collect the coordinates and attributes of surface points on Earth. The area acquisition targets the large areas of images which are geometrical and physical features. The mobility acquisition incorporates with GPS, RS, and GIS to observe the Earth system.

After gathering data, the main characteristics of spatial data which are size, complexity, dimension, and certainty definitely differentiate from the common data. The location-based spatial and no-spatial data imply the implication of objects. The spatial object is a core of an object-oriented data model such as point, line, area, or complex objects in the large dataset. A point can be a single, directed, and grouped point which presents the location on the Earth surface without shape and size in the computerized system. Then the interconnection between points presents a line as rivers and roads. The line presents the spatial curve of Earth surface as linear from point-to-point or the reticulated pattern. After that, the shape and size of lines illustrate an area as the curved surface of Earth. Finally, a complex object combines more than two objects of point, line, and area.

To enhance the efficiency of the target, three main characteristics of spatial data are time, location, and theme. Due to dynamic changes, time is sensitive to collect and compute the spatial data. The gathered data should be affirmative and reliable such as weather forecast. Next, the location determines the location relationship. The topology and relationship are demonstrated by position, azimuth, shape, and size of a spatial object. Then, the theme is an extraordinary characteristic which defines the land endurance, pH value, land cover, and population density. Some developed technology such as RS is necessary to request more thematic data acquisition. The spatial data is compulsory to utilize carefully because their characteristics impact the occurrence of spatial data fusion easily. However, the spatial data uncertainty is ubiquitous during the implementation. To maintain the consistency and the integration of spatial data for bringing out spatial objects, all techniques are required to solve those issues.

Spatial Data Model

A common use of spatial data models are hierarchical model, network model, relational model, and object-oriented model. Both hierarchical and network models address the relationship between objects which consider the path of data, storage, and accessibility. The hierarchical model of 1 or more than 1 root node(s) and leaf node(s) can store and access the hierarchical database

efficiently although users' interaction are inconvenient. The network model relationship is the connection between arbitrary records with others. Then, the occurrence of multiple relationships is possible to depict as an undirected graph even if it is the irregular structure. The relational model composes of row as an entity record and column as an entity attribute. When the model concerns the relationship between two connected dimensional tables with conditions, the unique and foreign keywords require. The unique keyword is a single or composited attribute for indicating the entity, while the foreign keyword is the relationship between entities. If relational model contains the Boolean logic and arithmetic rules, the Structured Query Language (SQL) has to manage the data. The Object-Oriented model has been highly valued in GIS application since late 1980s and early 1990s. The spatial objects relationship is expressed by the generalization, union, and aggregation. After that the model was integrated between the object-oriented and database techniques. When the complicated relationship or nested relationship exists, it is better to present as a table structure.

Spatial Databases

The spatial database is built for being a tool equipment and data manipulation. The definition, representation, and storage of spatial data diverge the common transactional data. The spatial data can be collected by the data manipulation of surveying and mapping database. The different purposes will be fulfilled with the different database. There are general database, graphic database, and image database. In addition, the digital elevation model (DEM) database is productive data arrangement. When a unified spatial index is developed, users can find the data of any domain fast. The large-scale databases are also not against the performance of data display. The spatial index allows users to access data by retrieving and transferring among different levels effortlessly.

From GGDI to Big Data

The GGDI which stands for Global Geospatial Data Infrastructure can run on the Digital Earth. GGDI to Big data is from 1970s which published the "Information Society." The study discussed on a global information infrastructure with the telecommunication infrastructure. After 10 years, the "Centralized Land Information Database" introduced a more complex network of distributed land information. The data resource management was adopted from a common approach to a computer based. Then in 1990s, the Spatial Data Infrastructure (SDI) was originated to support and accelerate the geographic data exchange standards. The local interests, demands, and constraints of country or organization are effected by the developed GGDI.

Presently, developers who usually apply the new technology for supporting their work, study and life, the digital contents are invented to present in form of internet of things and cloud computing. Thus, the big data seems to be an essential source for the digital content applications. The simulation of digital earth becomes popular in many countries. Most of the services utilize the technology of RS, GIS, ML, AL, and Internet of thing. The applications on network mapping, Web 2.0, and mobile sensors on vehicles impact significantly to the users' experience. Therefore, the spatial information system is obvious different from a traditional information system. The new geo-information systems are professional to serve the data and publish to all users. Hence, the new geo-information era improves the whole system of the geospatial information industry to prosperity. Soon the geo-information benefits for all aspects and leads to smart city definitely.

Key Research Findings

Bottleneck of SDM

Because of complex spatial entities, geographic location of features, boundaries, and relationship, SDM is capable to identify. However, the bottleneck of SDM is attentive. Some researches were

achieved with those goals but some difficulties need to be solved.

- Excessive Spatial Data

The geo-graphical information is imperative to accomplish the application development. The data source comes from RS, digital techniques, networks, multimedia, and images. So, the spatial database expands and grows rapidly. With the issues of immense dataset, the traditional implementation cannot manipulate with tons of spatial data such as the enumeration method of data analysis. Therefore, the data mining (DM) was introduced and capable to improve.

- High-Dimensional Spatial Data

The spatial data grows in both horizontal and vertical attributes, but the GIS information is still inadequate to illustrate the spatial data with high-dimension structure. The discovered knowledge can resolve the challenged. For example how to accelerate for searching and query the target data or how to decline the number of dimensional data.

- Polluted Spatial Data

The spatial entities may contain both useful and polluted data. The organized spatial data should concern about position accuracy, attribute accuracy, consistency, lineage, and integrity. The polluted spatial data is in a form of incompleteness, dynamic changes, noise, redundancy, and sparsity. They can cause a low-quality standard of data accuracy.

- Uncertain Spatial Data

The uncertain spatial data in SDM is impossible to avoid because of the approximate sample data and abstract mathematical models. Due to the crisp set, probability theory, GIS model, the sensitivity analysis for SDM can normalize to be a certain data. However, a related special entity of every single attribute can be emphasized instead of spatial data uncertainty.

- Mining Differences

The SDM utilization succeeds in understanding and gathering the different cognitive hierarchy of spatial dataset. The different mining on different aspects will return the different results from the results of discovered knowledge despite the same database. In addition, the making use of SDM can present a hierarchical decision-maker's awareness among those differences.

- Problems to Represent the Discovered Knowledge

The natural language is the best way to illustrate the conceptual knowledge which is a kind of the discovered knowledge. An exclusion of data uncertainty in SDM process will return incomplete knowledge and be full of errors although the adequate techniques are provided.

Methods and Techniques in SDM

The variety of methods and techniques impact directly the discovered knowledge in SDM. Each method supports the different characteristics based on the context. The SDM also concerns the evolution of human thinking network and optimal solution processes.

Crisp Set Theory

The Crisp Set Theory was introduced by Cantor in nineteenth century based on the modern mathematics. The theory approaches on probability, evidence theory, spatial statistics, spatial analysis, and data field. The probability which was conducted by Arthurs in 1965 works properly in the stochastic probabilities with randomness spatial data. Actually, the probability is unlike from the likelihood. The big probability does not present the high likelihood and vice versa.

Then, the evidence theory is known as Dempster-Shafer theory or significance theory. When the evidence exists, the hypothesis

evaluates a minimum of the belief function. Hence, the hypothesis cannot be denied because the plausibility function measures the maximum degree. However, the evidence is null by the unsupported interval. The evidence theory can also be an identification to probability theory. It implies that the evidence theory is the extension of the probability theory. The evidence theory is divided into two parts by the SDM technique which are the certainness measures confidence and the uncertainness measures likelihood. So, the framework of evidence data mining (EDM) concentrates on mass functions and mass operators to discover the knowledge.

Moreover, the spatial clustering presents the concept of similarity. The cluster determines the dataset to maximize the similarity and minimize the dissimilarity between clusters. The data is clustered by the object characteristics through spatial raw data. Only the basic object attributes can figure directly the meaningful clusters in a spatial dataset. Then, the data point in a multi-dimensional feature space can also cluster for the pattern recognition.

Extended Set Theory

SDM is capable on the similarity of human thinking on certainty and uncertainty. The crisp set examines the certainty, while the extended crisp set considers the uncertainty. To classify which group elements belong to, the crisp set bounds the binary logic of 0 and 1 as the full membership or no membership. However, the indeterminate boundaries are not accurate with the crisp set methods. Hence, SDM enhances the crisp set theory on uncertainty by using fuzzy set, rough set, and cloud model.

The fuzzy set is the fuzziness on a fuzzy membership. A close interval as [0, 1] of the partial membership defines an uncertainty probability. Two main fuzzy techniques are the fuzzy comprehensive evaluation and fuzzy clustering analysis. The fuzzy set is more proper to classify the spatial heterogeneous distribution of geographical uncertainty. The more fuzziness is created, the more difficult and complex system is. The approximation of the elements is close to 1 means the more possible belong to the target class.

Next, a rough set focuses on an incompleteness of the lower and upper approximations in SDM. An incompleteness-based reasoning method fulfills a decision-making by the characteristic of certainty and uncertainty data. The rough set favors on the partial membership with a set of an open interval and two different terminals as $\{0, (0, 1), 1\}$. The difference between lower approximation and upper approximation sets is the indeterminate boundary whether the sufficient information of objects can classify the class belongs to. The more information of the equivalent class is classified, the more accurate objects description is.

Finally, the cloud model requires the partial membership with the random close intervals. The data distributes stochastically in the space as random [0, 1]. The randomness integrates with the fuzziness in order to conduce a model of mutual information between qualitative concepts and quantitative data.

Bionic Method

The Bionic method is the representative of Artificial Neural Network (ANN) and genetic algorithm. The neural network simulates a human brain as the networked patterns by a self-adaptive nonlinear dynamic system. The ANN contains many neurons which are connected with the immense and sophisticated junctions. The classification, clustering, and prediction GIS data of SDM with the complicate system and connected plentiful of neurons are successful by ANN. The ANN implements more precisely than the symbolic classification. The numerous neurons processing seems as a network that creates the nonlinear function of a complex system. The information processing of neuron network involves with three layers which are input, middle, and output. The three layers operate depending on the dynamic response from the network status to the external data input. If between input and output layers have multiple hidden layers of units, ANN turns to be a deep neural network with the given method of deep learning. ANNs can reduce the noise disturbance in pattern recognition. So, the network is efficient on the high fault tolerance and robustness. If ANN is the nonlinear system

with many input variables, their convergence, stability, local minimum, and parameter adjustments of the network will be affected.

Others Related

- Rule Induction

The rule induction focuses on searching generic rule from the massive empirical data. The induction relates the basic statistical facts and instance of a big data while the deduction involves with axioms or basis reasoning on acknowledge knowledge (Clark and Niblet 1987). A concept tree and the rule induction which are the high-level patterns manipulate on uncovering the patterns among the spatial dataset.

- Decision Tree

The decision tree evolves the classification or decision set as a tree under certain spatial features. A test function on the tree structure of SDM generates the training object sets. Every created branch sets contain lower-nodes and sub-branches which generate iteratively. The leaf nodes are possible to be positive and negative examples.

- Visualization Techniques

In SDM, the spatial data and knowledge are better to explore and present visually. The form of visualization is worth than the words. The abstract patterns, relations, and tendencies of spatial data can be converted into the visualization by the computerized mechanism. The final results of visualization can be clarified by charts, maps, animations, graphics, images, trees, histograms, table, multimedia, and data cubes.

SDM Software

The SDM software implements the functions for discovering the spatial knowledge. The spatial data is truly existent the immense volume in the database. The implementation techniques are made use to instruct the computer understand what is knowledge to find and how to find the spatial knowledge. The SDM software has to manipulate with the data mining system. The decision-making system is supported by the extraction of high level information. The Knowledge Discovery and data mining are invented as a technique to reveal the strategic information hidden.

Not all tools can support the knowledge discovery and data mining, the proper tool will be selected based on the data characteristics. As a feature classification scheme, was purposed and studied by Goebel and Gruenwald (1999). The researchers extracted the features of 43 different tools based on 3 groups that are general characteristics, database connectivity, and data mining characteristics. The results showed that first 2 groups describe the ability of each tool supports what kind of data. Then the third group showed that rule induction, decision trees, and statistical methods are the standard data mining techniques while the fuzzy and rough sets or genetic algorithms started developing on the knowledge discovery software.

The software can interact with the database and data warehouse. After that, the distributed and heterogeneous data are needed to operate from offline-based to online-based (cloud computing). The technology of Google or Baidu have to accelerate the services for end-users which support the location-based, positioning, recommendation and manufacturing. The application started with toolkit that is ready-to-use software. Their functions provide the variety of data mining algorithms such as IBM Intelligent Miner, SPSS Clementine, SAS Enterprise Miner.

The SDM software for the spatial data mainly focuses on the GIS data and RS images. Li et al. (2016) studied and implemented the GISDBMiner for GIS data and the RSImageMiner for imagery data. The GISDBMiner supports the common structured data which is stored in the database. Users start by sending the knowledge discovery command and the signals to spatial DBMS which enable to access the spatial database. Then, the algorithm performs and returns the discovered knowledge as the output. For the RSImageMiner is powerful of

the data management and data mining on image features such as shape, color, texture and pattern. The RSImageMiner manipulates the RS imagery data which working on the mining process to discover the feature knowledge. Then, continue with the knowledge-based image classification, retrieval and object recognition (Li et al 2015).

Examples of Application

The methods and techniques in SDM are powerful to increase the quality of the discovered knowledge. As the mathematical foundation of those algorithms, the classification, clustering, and prediction on the spatial data mining are interesting. Koperski (1999) reviewed a two-step decision classification by first spatial predicates extraction with the relief algorithm of ML. Both spatial (which refers to the data of geographic location, size and shape objects on the planet) and non-spatial (which refers to a creation of text, image, multi-media data for linking with the spatial data to specific the location) (Diwakar 2013) predicate the integrated with knowledge of classification decision. Then, the regional classification rules with a fuzzy decision tree in an object-oriented spatial database are examined (Marsala and Bigolin 1998). Moreover, the rough set applications also apply the various fields of approximate reasoning, ML, AI, pattern recognition, and knowledge discovery. The further improvement of rough set can be continued with the studying of the field of geospatial information sciences such as geo rough space.

The developed SDM applications focus on the behavioral data in the system due to the continuous growth in volume of the spatial data. The Internet-based geographic information (Goodchild 2007) and location-based sensing services are important to update the spatial information. The spatial data is employed in the electrical maps, planning networks, land use construction, and protection on farmland. Hence, the making use of GPS, GIS, remote sensing, radar, infrared computerized tomography imaging, etc. can be the merging techniques to rise up the application performance.

Future Directions for Research

From the SDM concepts and techniques mentioned above, there are previous researches that studied and applied the spatial data on geographical information to solve problems of resources changes (Berger 2001), environment changes (Mannion 1995), and land-use allocation effect (Carsjens and Van Der Knaap 2002). Hence, the possible future directions for research can adjust to an agriculture aspect. The trend of agricultural product is advertent and proposed. Since the main occupation in the agricultural countries works on the cultivation which can crop yearly. The mass volume of agricultural products is continuous growth. Then, the economic trend of each country and the world in both short and long-term can be driven by those farm produce. The product's popularity is the major impact for forecasting the demand and supply in the real market. If the quantity of agricultural product can be predictable, it definitely benefits to various clients. In term of investor, they can consider on the investment. While in term of government, they can determine the policy direction to support the agriculturists.

Cross-References

- Big Data Analysis for Social Good
- Big Data Analysis Techniques
- Data Cleaning
- Spatial Data Integration

References

Berger T (2001) Agent-based spatial models applied to agriculture: a simulation tool for technology diffusion, resource use changes and policy analysis. Agric Econ 25(2–3):245–260

Carsjens GJ, Van Der Knaap W (2002) Strategic land-use allocation: dealing with spatial relationships and fragmentation of agriculture. Landsc Urban Plan 58(2):171–179

Clark P, Niblet TT (1987) The CN2 induction algorithm. Mach Learn J 3(4):261–283

Diwakar S (2013) Spatial vs non spatial. https://www.slideshare.net/SumantDiwakar/spatial-vs-non-spatial. Publish on: 14 Apr 2013
Ester M, Frommelt A, Kriegel HP, Sander J (2000) Spatial data mining: database primitives, algorithms and efficient DBMS support. Int J Data Min Knowl Discov 4(2):193–216
Goebel M and Gruenwald L (1999) A survey of data mining and knowledge discovery software tools. ACM SIGKDD explorations newsletter 1(1):20–33
Goodchild MF (2007) Citizens as voluntary sensors: spatial data infrastructure in the world of web 2.0. IJSDIR 2:24–32
Han JW, Kamber M, Pei J (2012) Data mining: concepts and techniques, 3rd edn. Morgan Kaufmann Publishers Inc., Burlington
Koperski K (1999) A progressive refinement approach to spatial data mining. PhD thesis, Simon Fraser University, British Columbia
Li DR, Cheng T (1994) KDG-knowledge discovery from GIS. In: Proceeding of the Canadian conference on GIS, Ottawa, pp 1001–1012
Li DY, Du Y (2007) Artificial intelligence with uncertainty. Chapman and Hall/CRC, London
Li D, Wang S, Li D (2015) Spatial data mining: theory and application. Springer, Berlin/Heidelberg
Li D, Wang S, Yuan H, Li D (2016) Software and applications of spatial data mining. Wiley Interdiscip Rev Data Min Knowl Disc 6(3):84–114
Mannion AM (1995) Agriculture and environmental change: temporal and spatial dimensions. Wiley, Chichester
Marsala C, Bigolin NM (1998) Spatial data mining with fuzzy decision trees. In: Ebecken NFF (ed) Data mining. WIT Press, Boston, pp 235–248
Piatetsky-shapiro G (1994) An overview of knowledge discovery in databases: recent progress and challenges. In: Ziarko Wojciech P (ed) Rough sets, fuzzy sets and knowledge discovery. Springer, London, pp 1–10

Spatial Graph Big Data

Shashi Shekhar and Jayant Gupta
Department of Computer Science, University of Minnesota, Minneapolis, MN, USA

Synonyms

Spatial networks big data; Spatiotemporal graphs big data; Spatiotemporal networks big data

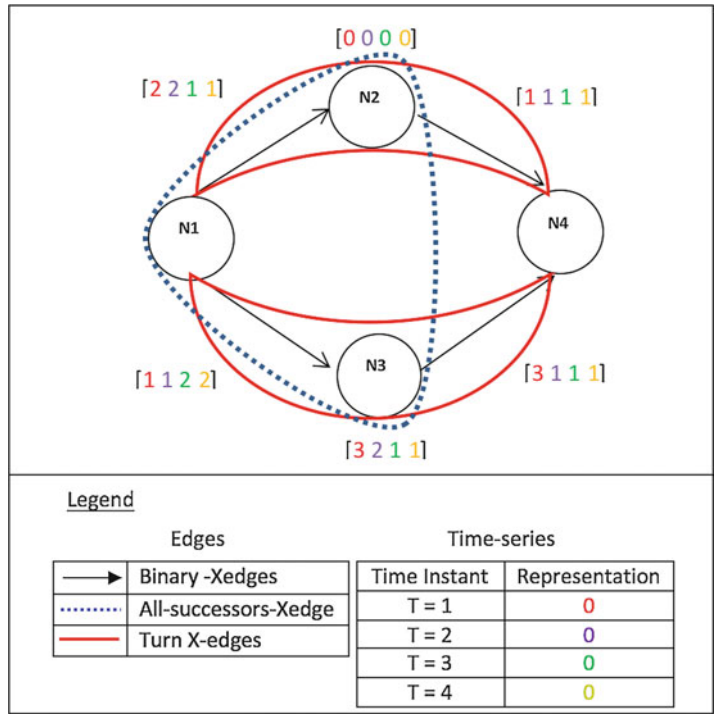
Definitions

Digital modeling of real-world networks (e.g., road networks, river network) to accurately represent geographic information is done using spatial graphs. Spatial graphs can represent n-ary relationships to model complex relations in the network. They differ from existing geographical models that can only represent binary relationships. Spatial graph is formally defined using the concepts of Xnodes and Xedges and Xgraphs in the following paragraph.

Xnode represents a real-world network feature (e.g., road intersection) that can have scalar or structured values (e.g., an array of scalars). **Xedge** is a tree of Xnodes that can have scalar or structural values. Further, Xedge can be classified based on the type of network features being modeled (e.g., turn-Xedges) as shown in Fig. 1. **Xgraph** can be defined as a set of Xnodes and a set of Xedges. **Spatial graph** is an ensemble of Xgraphs, which can represent n-ary relationships or discontinuity in a real-world network. Spatial graphs differ from hypergraphs because it consists of Xedge that can have a tree structure, whereas hypergraphs do not adhere to any specific structure.

Spatial graph big data is defined as a dataset with the following properties: (a) it describes the attributes of an Xgraph including attributes of its Xnodes, Xedges, and other substructures and (b) its volume, velocity, or variety exceeds the capacity of current data platforms. Examples include the longitudinal traffic volume associated with transportation networks such as road maps, as well as routes of airlines, buses, trains, ships, etc. as detailed in section "Examples of Application" on applications.

Figure 1 depicts a spatial graph big data with four nodes representing road intersections, namely, N1, N2, N3, and N4. The graph has seven Xedges including four binary Xedges (N1 → N2), (N2 → N4), (N1 → N3), and (N3 → N4) representing road segments connecting adjacent road intersections. The graph has an all-successor Xedge (N1 → (N2, N3)) to represent the group of binary edges (N1 → N2) and (N1 → N3) from Xnode N1 to its children Xnodes N2 and



Spatial Graph Big Data, Fig. 1 Example of spatial graph big data

N3. It also has two turn-Xedges (N1→N2→N4) representing a right turn and (N1→N3 →N4) representing a left turn. The time series associated with the turn-Xedges represent turn delays. In this example, the right turn (i.e., N1→N2→N4) has no wait. However, the left turn has a wait of 3 for start-time 1, 2 for start-time 2, 1 for start-time 3, and no wait for start-time 4. Note that, in the figure, there are seven Xedges but six time series, because all-successor Xedges do not have a time series associated with it. For brevity "spatial graph" term is used in place of "spatial graph big data" in the following text.

Overview

City development aims to connect existing resources to ensure a productive environment for prospective citizens. Historically, the urban development has been driven by the stability of the major roadways that form the backbone of the urban infrastructure (Strano et al. 2012). In modern times, urban dwellers and goods are increasingly connected by multiple networks supporting different means of transportation. The road networks have been augmented by rail networks (intercity, intracity), airport networks, and the port networks found located in cities near major rivers or navigable water bodies. These networks heavily influence many aspects of modern society, and they are implicated in many modern problems such as disease spread, congestion, and urban sprawl (Barthélemy 2011). Therefore, it is important to know how to model and analyze these types of transportation networks. In this regard, technology plays an important role to model the networks and understand their utilization.

Technological advancements, especially in the field of satellite systems, the Internet, and smartphones have had a major impact on the use and modeling of urban networks. GPS (Masumoto 1993) one of the Global Navigation Satellite System (GNSS) technologies (Lechner and Baumann 2000), is now used extensively for navigation on road networks. Further, Internet accessibility to smartphone users has been made possible by the advancement in electronics. Furthermore, when monitored at regular intervals, the location data from these devices allows us to digitally map

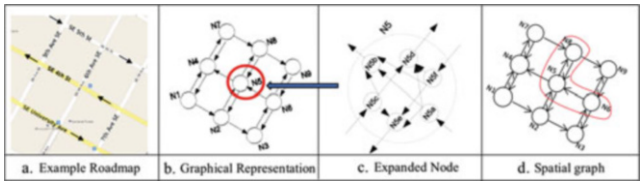
various movement activities (e.g., travel routes) to their physical locations, using map-matching techniques (Jensen and Tradišauskas 2009) for every user to perform different types of analysis (e.g., generating historical profiles of driver behavior).

Proliferation of technologies and the location data they generate is growing exponentially. In 2015, there were already around 2.6 billion smartphone subscriptions globally, and this number will continue to rise in the future especially in developing countries where the market is still maturing (Lunden 2015). In addition, commercial vehicles are using more sophisticated GPS devices that can monitor a variety of engine parameters (e.g., fuel usage, speed, etc.) in addition to location. Therefore, it is necessary to build technologies which can accurately and effectively model location-based data that are ever-growing size, variety, and update rate. In a big data paradigm, these issues correspond to the standard three Vs, i.e., volume, velocity, and variety (Zikopoulos and Eaton 2011). New database management systems are required to effectively harness these datasets.

In general, database applications are modeled using a three-step design process beginning with a conceptual data model (e.g., entity-relationship model), a logical data model (e.g., relational schema), and finally a physical design (e.g., R-Tree). To handle spatial features, the entity-relationship (ER) models were modified to pictogram-enhanced ER (PEER) to improve the geographical representation of the data in the model. The relational schema was modified by incorporating appropriate data types from SQL3 onward and translating the spatial relationship into spatial integrity constraints (Shekhar et al. 2004). Spatial graphs represent the physical modeling phase for modeling spatial network data, where, traditional graphs were modified to become a spatial graph to represent complex network properties with scalar or structural values.

To illustrate the limitations of simple graphs while modeling turns, consider the example road map as shown in Fig. 2a. Figure 2b shows a graphical representation of the road map where every node represents a road intersection and every directed edge represents a left or right lane of the road segment. In the original graph, the right turn at node N5 can be modeled as N6 –> N5 –> N8, which requires the use of three nodes and two edges (3N, 2E). Consider, Fig. 2c, where the node N5 is expanded to six sub-nodes to model the turns at the intersection. Now the turn can be represented using N5a –> N5e, i.e., with just two sub-nodes and one edge, and no information is required from the other intersections. It is useful to be able to model turns at an intersection independent of other intersections. Finally, consider Fig. 2d that shows spatial graphs of segments and turns as turn-Xedges to represent the road map. The red-colored Xedge represents the turn; thus, with spatial graph, the turns can be modeled using a single Xedge.

Spatial graphs have been extensively studied in three main areas, namely, data modeling, database storage, and data mining algorithms. The research in spatial graph modeling is motivated to develop simple and effective representations of the data. For example, network size can grow rapidly as new nodes are added, and a simple model should easily accommodate and adapt to the changes. The spatial graph research in data storage is motivated to provide efficient graph-based operations (e.g., node retrieval, edge retrieval). The operations require data related to the same route; therefore, techniques (Evans et al. 2010) that store data on routes together helps to reduce the number of data page access. The research in the area of data mining uses spatial models to solve problems such as finding shortest



Spatial Graph Big Data, Fig. 2 Modeling turns (Shekhar et al. 2012) [Best in Color]

cost (e.g., shortest travel time) to different destinations or route planning to maximize traffic flow or crowd movement toward a destination (George et al. 2007). Section "Research Areas" describes each of these domains in further detail. Readers interested in example applications or future directions may consult section "Examples of Application" or section "Future Directions of Research" of this entry, respectively.

Research Areas

Modeling Spatial Graphs

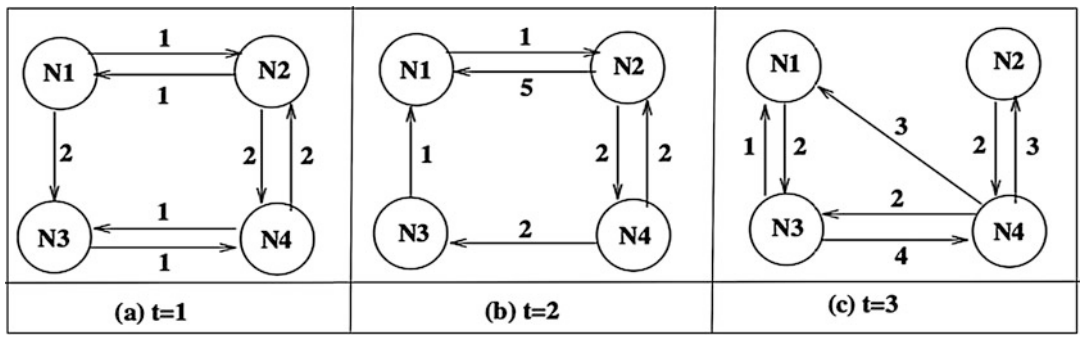
Spatial graphs typically represent transportation networks that can be viewed from one of two frames of reference, Eulerian or Lagrangian (Batchelor 2000). In the Eulerian frame of reference, traffic is observed as it travels past specific locations in the space over a period of time. It is similar to sitting on the side of a highway and watching the traffic pass a fixed location. In the Lagrangian frame of reference, the observer follows an individual moving object as it moves through space and time. This can be visualized as sitting in a car and driving down a highway. The following text discusses three spatial graph models: the snapshot and the time-aggregated graph (TAG) models, which use the Eulerian frame of reference, and the time-expanded graph (TEG) model, which uses the Lagrangian frame of reference (George et al. 2007).

1. **Snapshot Model**

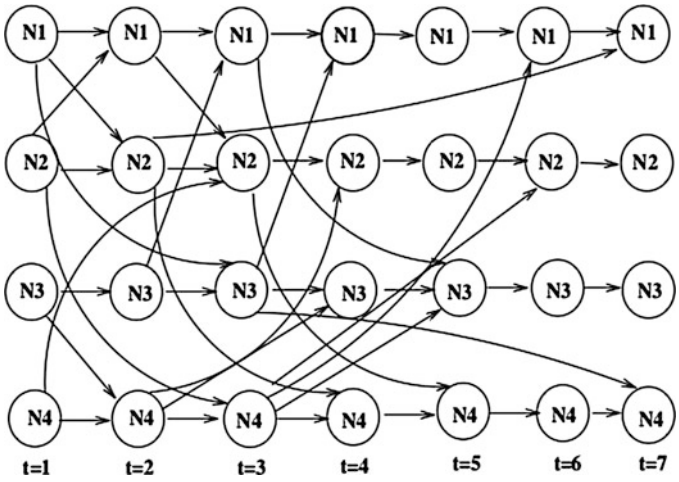
A snapshot model represents a spatial graph with temporal attributes using a finite set of nodes and a finite set of edges connecting the nodes. Temporal attributes are represented by discrete time steps, each represented by a snapshot as shown in Fig. 9. The figure shows three time steps, $t = 1$ to $t = 3$, and shows the effect of temporal change on an edge, indicating travel time. For example, at time $t = 1$, edge N2-N1 has a value of 1. In the next time step, $t = 2$, the edge value increases to 5, indicating an increased travel cost for the edge.

2. **Time-Expanded Graphs**

A time-expanded graph (TEG) (Köhler et al. 2002) model replicates each node along the time series such that a time-varying attribute (e.g., travel time) is represented between replicated nodes. Figure 4 shows the TEG corresponding to the spatial graph displayed earlier (Fig. 3). From the figure, observe that for edge N1-N2, weight of 1 is represented by an edge from N1 at $t = 1$ to N2 at $t = 2$. Further, the recurrence of similar edge across time between N1 and N2 at $t = 2$ mimics the invariance of edge weight over this time period. It is also interesting to see that N2 at $t = 1$ has an edge to N1 at $t = 2$ and that N2 at $t = 2$ has an edge to N1 at $t = 7$. This is because the edge weight for N2-N1 changes from 1 at $t = 1$ to 5 at $t = 2$ (Fig. 4).



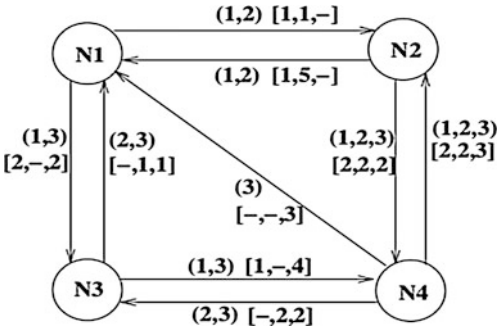
Spatial Graph Big Data, Fig. 3 Snapshot model (George et al. 2007)



Spatial Graph Big Data, Fig. 4 Time-expanded graph (George et al. 2007)

3. Time-Aggregated Graphs

A time-aggregated graph (TAG) stores a spatial graph with temporal attributes as the tuple {**N, E, Map(N, TS_N), Map(E,TS_E), W**}, where N is a set of nodes, E is a set of edges (E), Map(N, TS_N) is the mappings from nodes to the time series associated with the nodes, Map(E,TS_E) is the mappings from edges to the time series associated with the edges, and W is the time-dependent weights (e.g., travel time). Figure 5 shows the corresponding TAG for the networks as shown in Figs. 3 and 4. Observe that that node N1 has two time series associated with it for edge N1-N2 (1, 2) and N1-N3 (1, 3). Then, observe that the time-dependent weights for these edges are [1,1, −] and [2, −, 2], respectively. The "-" symbol represents the time $t = 3$ when the edge N1-N2 does not exist.

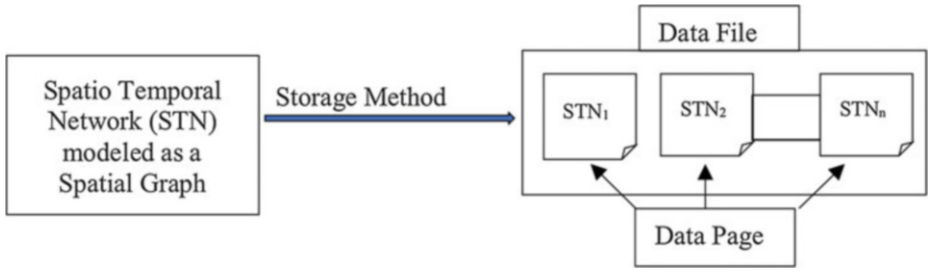


Spatial Graph Big Data, Fig. 5 Time-aggregated graph (George et al. 2007)

Storage of Spatial Graphs

Large-scale spatial graphs with temporal attribute have hundreds of thousands of nodes and millions of time steps, amounting to terabytes of data. It is appropriate therefore to focus on secondary techniques for the storing of time-attributed spatial graphs for database systems. Figure 6 shows the process of storing a spatial graph. These methods make use of data files consisting of data pages and an indexing method. These methods employ an indexing method to output a data file containing the spatial graphs partitioned across a set of data pages. Incorporating temporal data into a spatial graph poses significant challenges to their storage and analysis of disk I/O. Network topology and temporal access patterns add further constraints to the accessibility of data records.

One solution is to physically store topologically related nodes in the same data page, reducing the cost of data-page retrieval. Further, as the storage size decreases, the I/O cost is reduced due to a fewer number of pages containing the same data. The following texts describe three methods for storing spatial graphs: snapshot partitioning, longitudinal partitioning, and non-orthogonal partitioning. First, however, it is necessary to explain the concept of orthogonal storage vs. non-orthogonal storage, also known as synchronous time grouping vs. asynchronous time grouping.



Spatial Graph Big Data, Fig. 6 Process of storing spatiotemporal networks (STN)

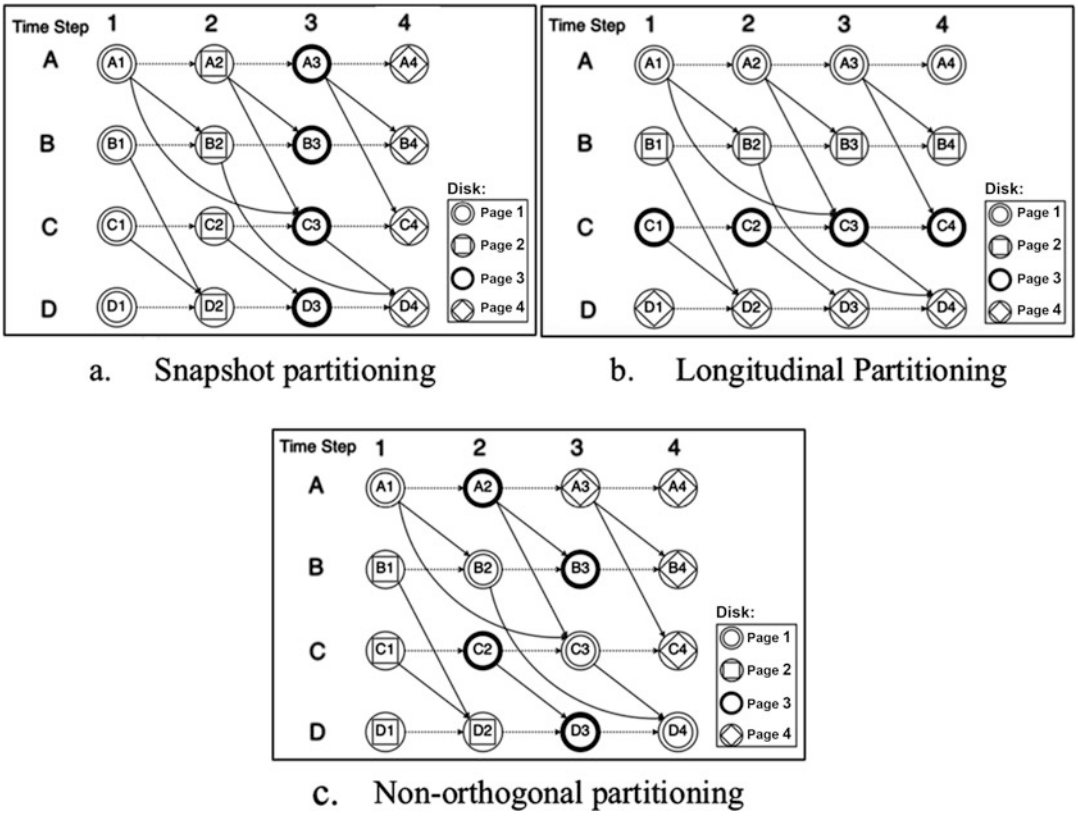
Synchronous vs. Asynchronous Time Grouping
Synchronous time grouping is the clustering of data within a set time series, whereby some numbers of nodes and edges are stored within a set time interval. Each page is a snapshot, and longitudinal scheme represents a continuous time interval, either one time step or the entire time series. An example is storage of A1, B1, C1, and D1 in the same data page for snapshot partitioning as shown in Fig. 7a and storage of A1, A2, A3, and A4 in the same data page for longitudinal partitioning as shown in Fig. 7b. Asynchronous time data grouping allows storing data for disjoint time intervals. Thus, data from different time intervals are grouped and stored on the same data page. This model is known as sub-node model. It is useful for queries that retrieve the traversal times beginning at each time instant where temporal data is mostly not accessible orthogonally. An example is storage of A1, B2, C3, and D4 in the same data page as shown in Fig. 7c for Lagrangian-connectivity partitioning.

1. **Snapshot Partitioning**

Snapshot partitioning stores the data in pages using quad tree-based geometrical structures (e.g., R-Trees, R+−Trees, etc.). Figure 7a shows an example of snapshot partitioning, where a spatial graph with temporal attributes is represented as a snapshot model partitioned across the disk pages. As can be seen, the model preserves the graph state for a given time instant (say time $t = 1$). Further, the graph states are strictly partitioned based on time; thus, when traversing through the graph across time, multiple-disk access needs to take place. For example, if a car's route has to be evaluated from A to D via C starting from time $t = 1$, it will require an access to edge AC at $t = 1$ stored in data page 1 and then access to edge CD at $t = 3$ stored in data page 3. Thus, the evaluation requires access to two data pages.

2. **Longitudinal Partitioning**

Longitudinal partitioning stores spatial graphs with temporal attributes based on the storage structure of the adjacency-list main memory. Figure 7b shows the longitudinal storage. As can be seen, each node is stored with its attribute information and all outgoing edges with their attribute information as an adjacency list. This orthogonal storage solution suffers from the increasing disk I/O to evaluate routes in large spatiotemporal networks. The example network has a short time series compared to its graph size, allowing multiple node records (with adjacency list) to fit inside a data page. However, if the time-series length was larger, then all the nodes may need multiple pages for storage. This is due to the long time series being stored with each node, resulting in only a small number of storable nodes on each data page. Consider the evaluation of the route ACD using longitudinal partitioning. First it requires accessing the traversal time attribute of edge AC at $t = 1$, stored on data page 1. Then, it requires accessing edge CD at $t = 3$, stored on data page 3, to complete the evaluation. Thus, for longitudinal partitioning, the evaluation requires two data pages.



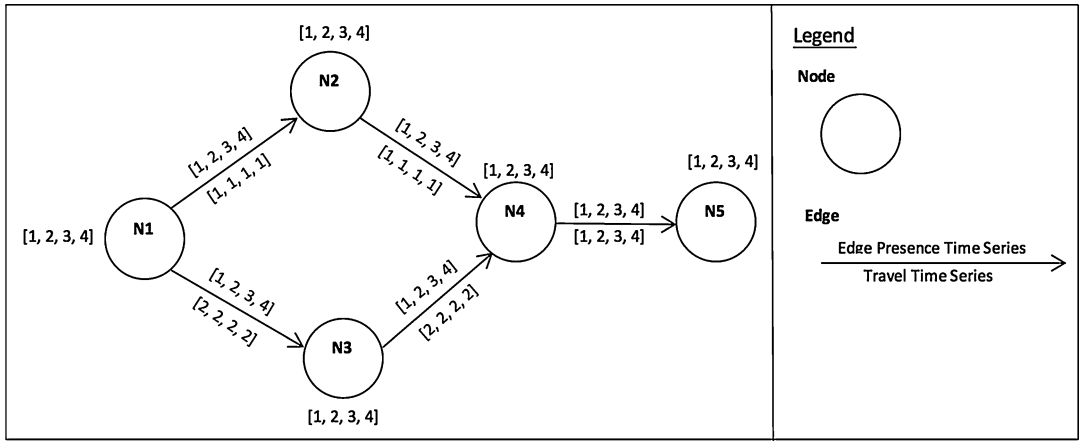
Spatial Graph Big Data, Fig. 7 Spatial graph storage techniques (Evans et al. 2010)

3. **Lagrangian-Connectivity Partitioning**

Lagrangian-connectivity partitioning (LCP) uses time-expanded graphs (TEG) to store spatial graphs with temporal attributes. Each partition stores the non-orthogonal patterns of route evaluation operations along with a novel data record based on sub-nodes. By representing a spatiotemporal network as a modified time-expanded graph, focusing on the Lagrangian connections between nodes (movement edges), a min-cut graph partitioning algorithm creates partitions clustering nodes by minimizing the cuts of these movement edges. This allows the algorithm to collocate connected temporal nodes together on data pages, stored as sub-node records as shown in Fig. 7c. This further reduces the I/O cost for the operations (described earlier). This can be seen by looking again at the evaluation of route ACD. Traversing from node A1 to C3 and then C3 to D4 requires only one data page as all relevant sub-node records are on the same data page.

Data Mining Algorithms on Spatial Graphs

Spatial graphs with temporal attributes are used in emergency traffic planning and route-finding services. These applications require routes with the smallest duration for a given start time (useful for known events) and the route with the smallest duration that can occur at any of the start time. Developing efficient algorithms for finding such routes in a time-varying spatial network is challenging because these journeys do not always display a greedy property or optimal substructure, making techniques like dynamic programming inapplicable. The following text will outline algorithms for finding a route with the smallest duration for a given start time followed by an algorithm that finds routes with the smallest duration that can occur at any of the start times.



Spatial Graph Big Data, Fig. 8 Example spatial graph with temporal attributes

1. **Shortest Path Computation for Time-Aggregated Graphs (SP-TAG)**

The SP-TAG algorithm (George et al. 2007) uses the time-aggregated graphs to represent spatial networks, where each node (road intersection) has a node presence time series and each edge (road segment) has an edge presence time series annotated with travel time. The algorithm keeps track of the earliest time a node can be reached and updates the values as optimal values are found. The algorithm assigns t_{start} to the first node and puts the node in a priority queue (Cormen 2009). The steps are repeated until the priority queue is empty. At each step, the lowest-cost node is extracted from the queue. Then, the values for all its adjacent nodes are updated, and the extracted node is marked as closed. For each update, the minimum between the previous value and the new value of *current time* + *traversal time* is chosen. Then, the adjacent node is added to the queue. The time complexity of the SP-TAG algorithm is O(m(logT+logn)), where T is the number of nodes and m is the number of edges in the time-aggregated graph.

Figure 8 shows an example spatial graph having nodes with node presence time series and edges with edge presence time series, travel time series. Table 1 shows the SP-TAG algorithm trace on this graph, which shows the computation of the shortest path from N1 to N5. Each row represents an iteration of the algorithm that updates the values to newly found optimal values. For example, at iteration 2, node N2 is closed, and the N4 gets updated as shown in iteration 3.

Spatial Graph Big Data, Table 1 Trace of the SP-TAG algorithm (George et al. 2007)

Iteration	N1	N2	N3	N4	N5
1	1 (closed)	∞	∞	∞	∞
2	1	2 (closed)	3	∞	∞
3	1	2	3 (closed)	3	∞
4	1	2	3	3 (closed)	6
5	1	2	3	3	6 (closed)

2. **Best Start Time Shortest Path (BEST) Algorithm**

The BEST algorithm uses time-aggregated graphs to represent the network, where every node has a node presence time series and every edge has an edge presence time series and travel time series. In the output, each node has a time series, with the ith entry representing the current, least travel time to the destination node for the start time t_i. The algorithm uses an iterative label correcting approach (Ahuja et al. 1993), and each

entry in a node time series is modified according to the following condition:

$C_u[t] = \text{minimum}\{C_u[t], \sigma_{uv}(t) + C[t + \sigma_{uv}(t)]\}$, where $uv \in E$,

$C_u[t]$ – Travel time from $u \in N$ to the destination for the start time t.

$\sigma_{uv}(t)$ – Travel time of the edge uv at time t.

The algorithm maintains a list of all nodes that change its cost according to the condition and terminates when there is no further improvement indicated by an empty list. The algorithm uses two-Q-based implementation (Pallottino 1984). The computational complexity of the BEST algorithm is $O(n^2mT)$, where n is the number of nodes, m is the number of edges, and T is the length of the time series.

Figure 9 shows an example with a figurative trace. At each step, the distance list from destination gets updated to show the shortest distance for all the time instants. Further, the parent pointer list gets updated to show the corresponding parent. Table 2 shows the trace for the algorithm showing shortest distance from N4 (destination) to all the nodes for each of the time instance. Further, the table shows the queue state. Note that in Fig. 9 edge presence time series and node presence time series are omitted because all the edges and nodes are present at all times.

Examples of Application

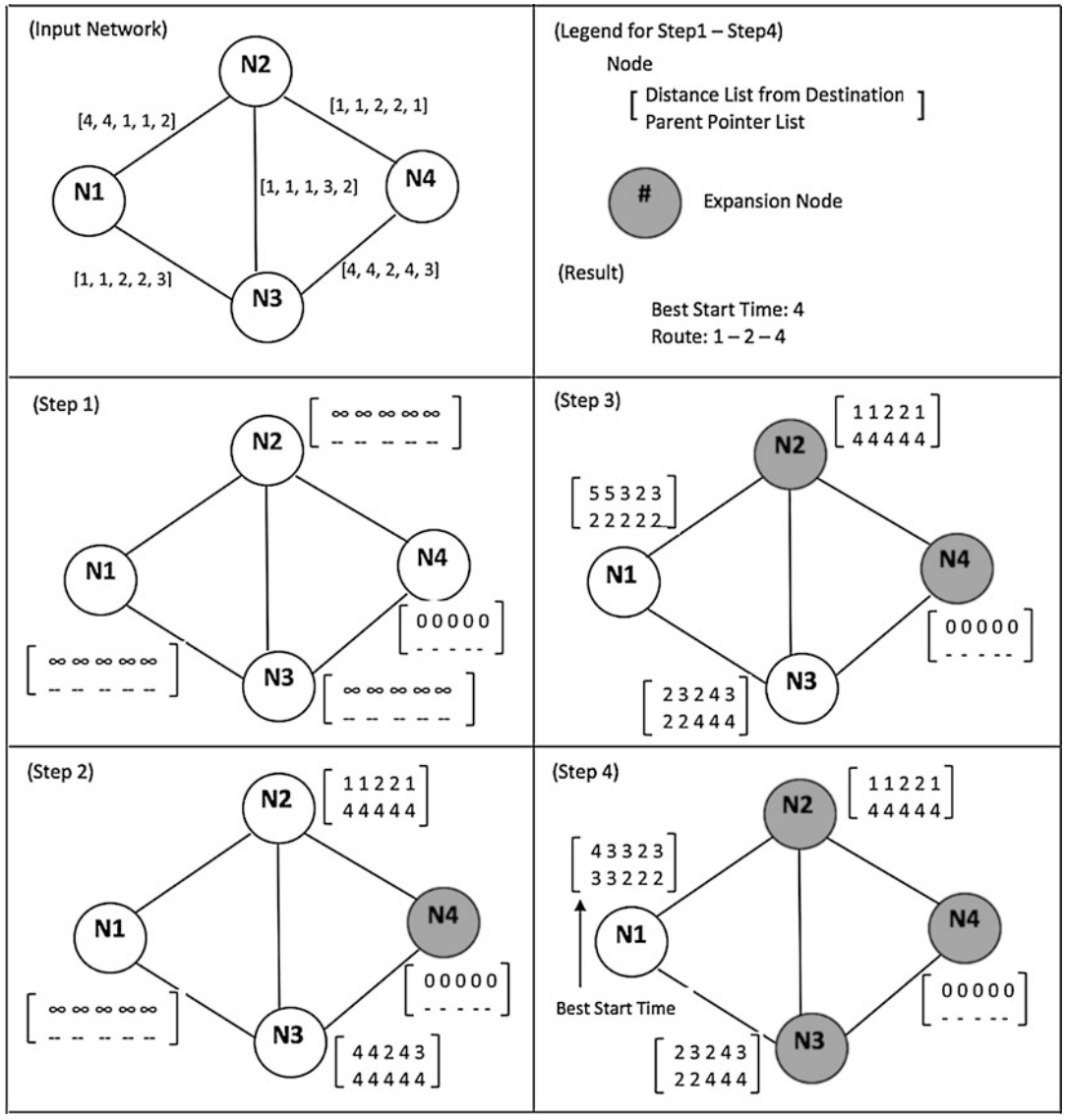
This section elaborates on three existing applications that rely on spatial graphs. Two of the applications show the use of different frames of reference (described in section "Modeling Spatial Graphs") to the real-world applications. Section "Historical Speed Profiles for Roadways" describes the use of vehicle trajectory from a Eulerian frame of reference, and section "GPS Trace Data for Improving Fuel Efficiency" describes the use of vehicle trajectory from a Lagrangian frame of reference. Section "Evacuation Route Planning" discusses an important societal problem that talks about the use of spatial graphs in disaster management.

Historical Speed Profiles for Roadways

Traditionally, digital road maps have consisted of center lines and topologies of the road networks. These maps are used by navigation devices and web VTEQ, probe vehicles, and highway sensors to compile travel time information across road segments for all times of the day and week at fine temporal resolution (seconds or minutes). The profiles have data for every 5 min, which can then be applied to the road segment, building up an accurate picture of speeds based on historical data. An example distribution is shown in Fig. 10 (Shekhar et al. 2012). Such temporally detailed (TD) road maps contain much more speed information than traditional road maps. While traditional road maps have only one scalar value of speed for a given road segment (e.g., EID 1), TD road maps may potentially list speed/travel time for a road segment (e.g., EID 1) for thousands of time points (Fig. 10a) in a typical week. This allows a commuter to compare alternate start times in addition to alternate routes. It may even allow comparison of (different) start time and route combinations to select distinct preferred routes and distinct start times. For example, route ranking may differ across rush hour and non-rush hour and in general across different start times. However, TD road maps are big, and their size may exceed 10^{13} items per year for the 100 million road segments in the USA when associated with per-minute values for speed or travel time. Thus, industry is using speed profiles that are a lossy compression based on the idea of a typical day of a week, as illustrated in Fig. 10b, where each road segment and day of the week pair is associated with a time series of speed values for each hour of the day.

GPS Trace Data for Improving Fuel Efficiency

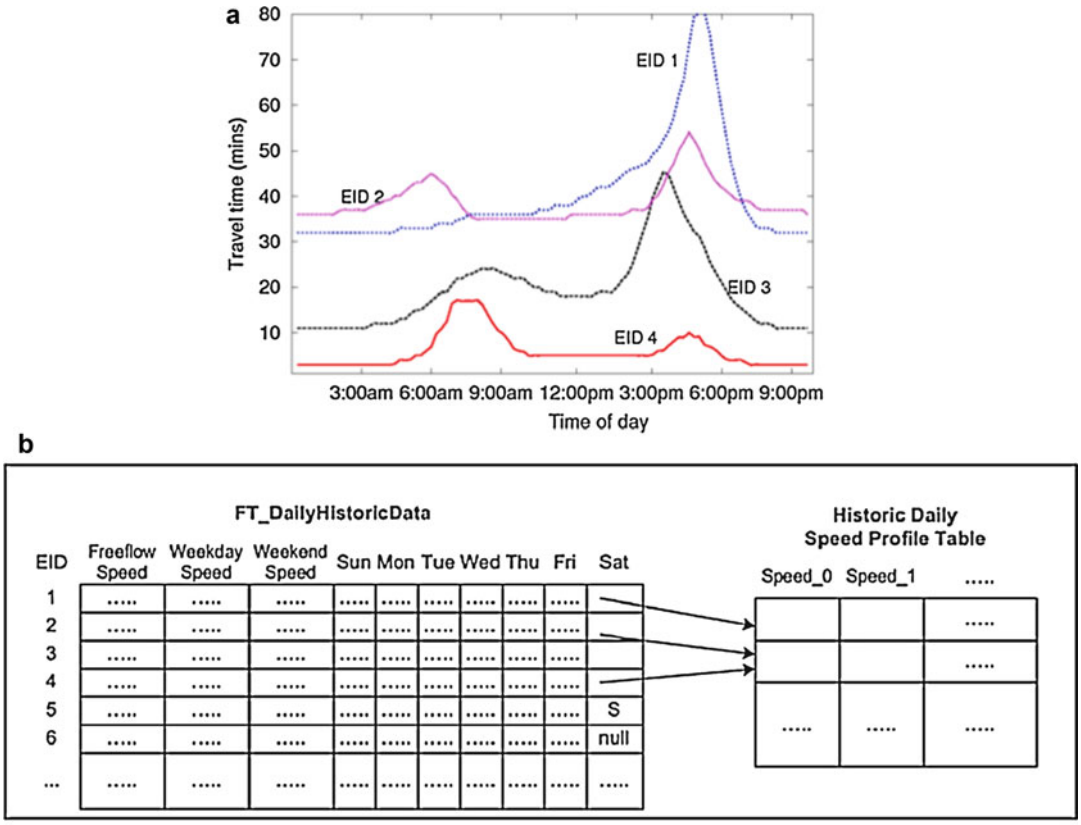
GPS trajectories are available for a large collection of vehicles due to rapid proliferation of cell phones, in-vehicle navigation devices, and other GPS data-logging devices such as those distributed by insurance companies. According to Shekhar et al. (2012), GPS traces allow indirect estimation of fuel efficiency and GHG emissions via estimation of vehicle speed, idling, and con-



Spatial Graph Big Data, Fig. 9 Trace of the BEST algorithm (George et al. 2007)

Spatial Graph Big Data, Table 2 Trace of the BEST algorithm (George et al. 2007)

Iteration	N1	N2	N3	N4	Queue
1	∞ ... ∞	∞ ... ∞	∞ ... ∞	[0, 0, 0, 0, 0]	N1
2	∞ ... ∞	[1, 1, 2, 2, 1]	[4, 4, 2, 4, 3]	[0, 0, 0, 0, 0]	N2, N3
3	∞ ... ∞	[1, 1, 2, 2, 1]	[2, 3, 2, 4, 3]	[0, 0, 0, 0, 0]	N3
4	[4, 3, 3, 2, 3]	[1, 1, 2, 2, 1]	[2, 3, 2, 4, 3]	[0, 0, 0, 0, 0]	N1
5	[4, 3, 3, 2, 3]	[1, 1, 2, 2, 1]	[2, 3, 2, 4, 3]	[0, 0, 0, 0, 0]	–

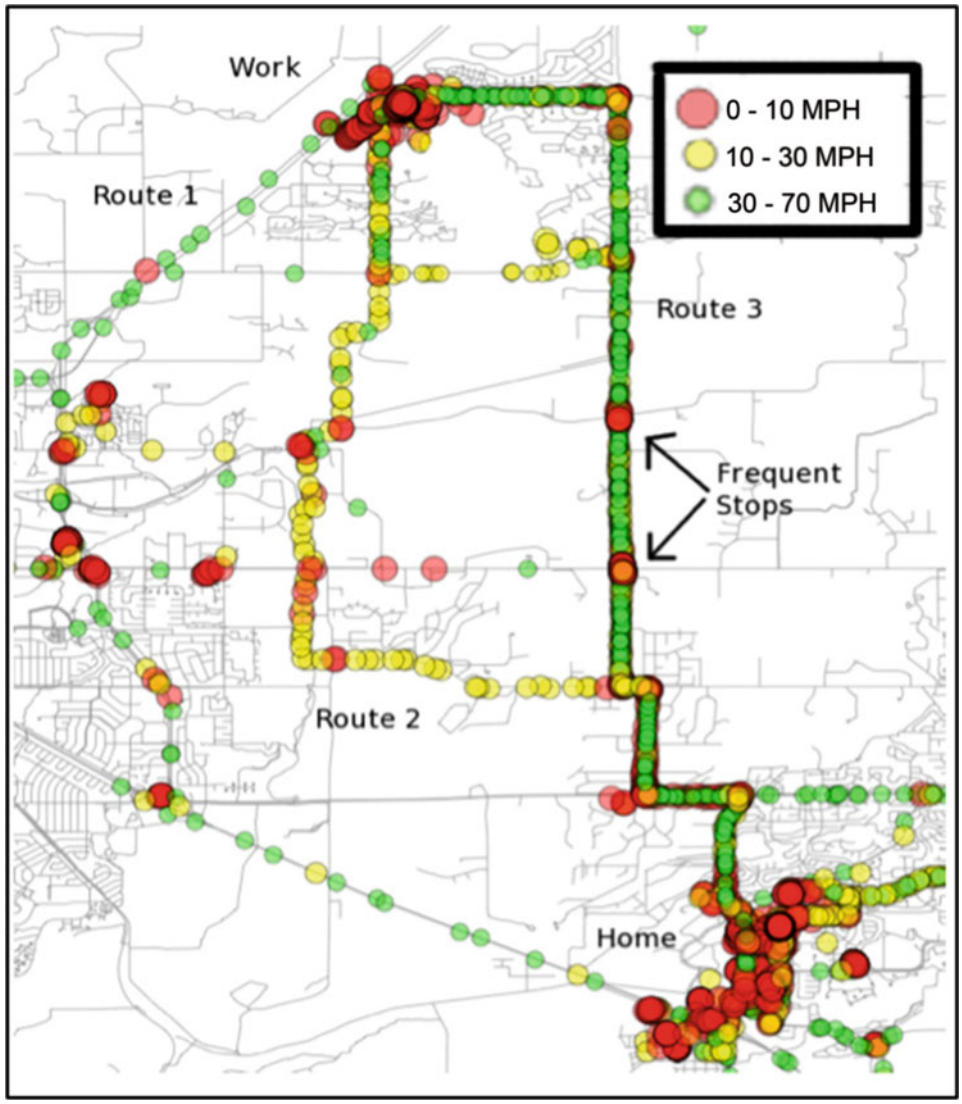


Spatial Graph Big Data, Fig. 10 Spatial big data on historical speed profiles (Shekhar et al. 2012)

gestion. They also make it possible to provide personalized route suggestions to users to reduce fuel consumption and GHG emissions. For example, Fig. 11 shows 3 months of GPS trace data from a commuter with each point representing a GPS record taken at 1 min intervals, 24 h a day, 7 days a week. As can be seen, three alternative commute routes are identified between home and work from this dataset. These routes can be compared for engine idling which are represented by darker (red) circles. Assuming the availability of a model to estimate fuel consumption from speed profiles, one may even rank alternative routes for fuel efficiency. Again, a key hurdle is the dataset size, which can reach 10^{1313} items per year given constant minute-resolution measurements for all 100 million US vehicles.

Evacuation Route Planning

Large public gatherings require effective management to preserve public safety. One of the key responsibilities of a civil administration is the capability to manage adverse conditions (e.g., terrorist acts, accidents) that may require finding efficient routes to evacuate all or a portion of a population. The evacuation route planning problem (Yang et al. 2012) finds routes that minimize evacuation time given a transportation network, a population, and a set of observations. To be effective and timely, evacuation planning methods need to adhere to network capacity constraints and provide reasonable computation time. Evacuation route planning involves (taking) a big data perspective due to the size of transportation networks (order



Spatial Graph Big Data, Fig. 11 A commuter's GPS tracks over 3 months reveal preferred routes (Shekhar et al. 2012)

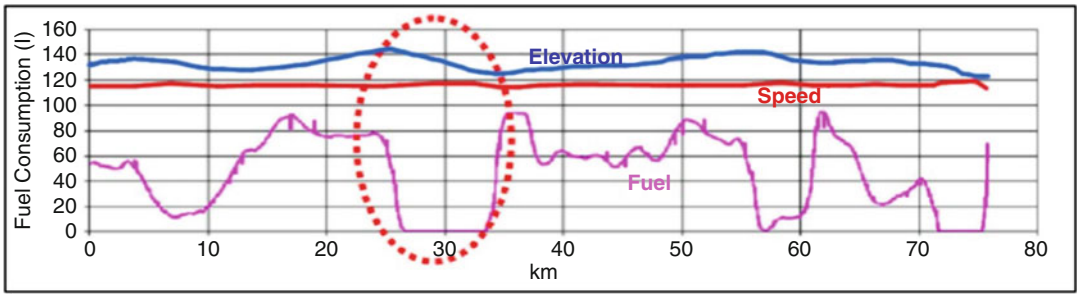
of 10^3 nodes and 10^6 edges), the large number of evacuees (order of 10^6), as well as the capacity constraints for every node and edge.

Future Directions of Research

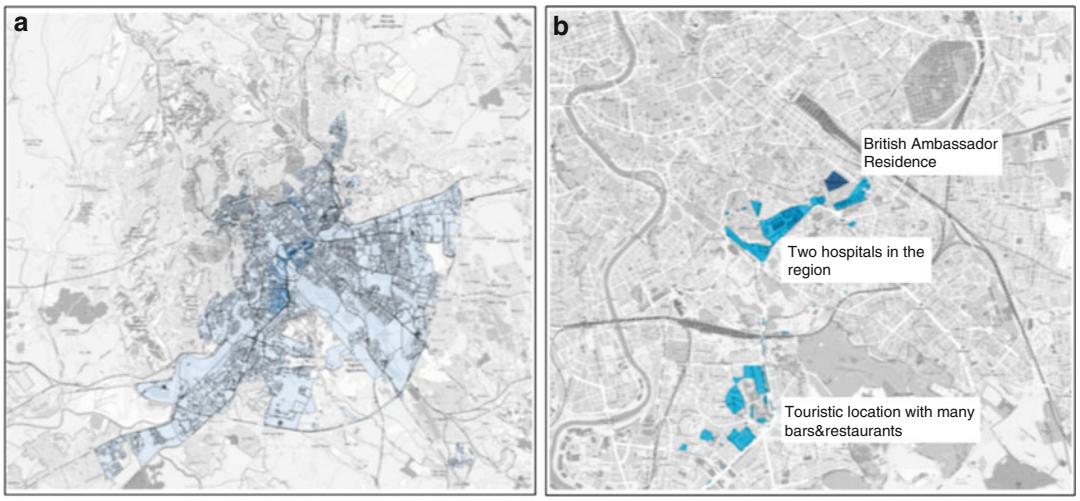
The section describes two future directions of research related to spatial graphs. The first is driven by the increased use of sensors that can measure every aspect of vehicles including its surroundings. This would potentially allow to make variety of profiles for a vehicle. The second is driven by the need to identify city areas that are affected due to multiple reasons (e.g., natural calamities, adverse neighborhood) based on trajectory data.

Time Annotated Engine Measurement Spatial Data

Engine measurement datasets may be used to study the impacts of the environment (e.g., ele-



Spatial Graph Big Data, Fig. 12 Engine measurement data improve understanding of fuel consumption (Capps et al. 2008)



Spatial Graph Big Data, Fig. 13 Case study on identification (Eftelioglu et al. 2018)

vation changes, weather), vehicles (e.g., weight, engine size, energy source), traffic management systems (e.g., traffic light timing policies), and driver behaviors (e.g., gentle acceleration or braking) on fuel savings and greenhouse gas emissions. Fuel efficiency can be estimated from fuel levels and distance traveled as well as engine idling from engine RPM. These attributes may be compared with geographic contexts such as (e.g., elevation changes) to improve understanding of fuel efficiency and greenhouse gas emission.

For example, Fig. 12 shows heavy truck fuel consumption as a function of elevation from a study (Capps et al. 2008) at Oak Ridge National Laboratory. Notice how drastically fuel consumption changes drastically with elevation slope changes. Commercial fleet owners have studied such datasets to fine-tune routes to reduce unnecessary idling. It is tantalizing to explore the potential of these datasets to help consumers gain similar fuel savings and reductions in greenhouse gas emission. However, these datasets can grow big. Measurements of 10 engine variables, once a minute, over the 100 million US vehicles in existence, may have 10^{14} data items per year.

Distressed Area Identification

Distressed area identification uses GPS trajectories on a road network to identify distressed areas or the regions that are usually avoided by the commuters. This emerging area is important to applications such as sociology, city/transporta-

tion planning, and crime mitigation, where it can help domain users to understand the driver behavior under varying adverse conditions (e.g., rush hour, congestion, dangerous neighborhoods, etc.).

For example, Fig. 13 shows results from a recent case study (Eftelioglu et al. 2018) on a real dataset consisting of 1312 vehicle GPS trajectories in Italy over a period of 3 years. The study used the input trajectory data and road network data to output interesting avoidance regions as shown in Fig. 13a. The zoomed-in view (Fig. 13b) shows that drivers avoid areas with increased security measures (e.g., important government buildings) or increased congestion (e.g., hospitals or tourist locations).

Cross-References

▸ Spatial Data Mining

References

Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows: theory, algorithms, and applications. Pearson. ISBN-13: 978-0136175490

Barthélemy M (2011) Spatial networks. Phys Rep 499(1):1–101. Elsevier

Batchelor GK (2000) An introduction to fluid dynamics. Cambridge University Press. ISBN 978–0521663960

Capps G, Franzese O, Knee B, Lascurain MB, Otaduy P (2008) Class-8 heavy truck duty cycle project final report. ORNL/TM-2008/122

Cormen TH (2009) Introduction to algorithms. MIT press. ISBN 978–8120340077. eMarketer, https://goo.gl/TcMzQX

Eftelioglu E, Tang X, Shekhar S (2018) Avoidance region discovery: a summary of results. SIAM international conference on data mining (Accepted)

Evans MR, Yang K, Kang JM, Shekhar S (2010) A lagrangian approach for storage of spatio-temporal network datasets: a summary of results. In: Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems. ACM, New York, pp 212–221

George B, Kim S, Shekhar S (2007) Spatio-temporal Network Databases and Routing Algorithms: A Summary of Results. In: Papadias D, Zhang D, Kollios G (eds) Advances in Spatial and Temporal Databases. SSTD 2007. Lecture Notes in Computer Science, vol 4605. Springer, Berlin/Heidelberg

Jensen C, Tradišauskas N (2009) Map matching. In: Liu L, Özsu MT (eds) Encyclopedia of database systems. Springer, Boston

Köhler E, Langkau K, Skutella M (2002) Time-expanded graphs for flow-dependent transit times. In: Möhring R, Raman R (eds) Algorithms – ESA 2002. Lecture notes in computer science, vol 2461. Springer, Berlin/Heidelberg

Lechner, W., Baumann, S. (2000). Global navigation satellite systems. Comput Electron Agric, 25(1), 67–85. Elsevier

Lunden I (2015) 6.1B Smartphone Users Globally By 2020. Overtaking basic fixed phone subscriptions. https://goo.gl/lcMcPK

Masumoto Y Global Positioning System (1993) U.S. Patent No. 5,210,540. U.S. Patent and Trademark Office, Washington, DC

Pallottino S (1984) Shortest-path methods: complexity, interrelations and new propositions. Networks 14(2):257–267. John Wiley & Sons, Inc

Shekhar S, Vatsavai RR, Ma X, Yoo JS (2004) Navigation systems: A spatial database perspective. as Chapter 3 in Location-Based Services, Agnes Voisard and Jochen Schiller. Elsevier, pp 41–80. ISBN 9781558609297

Shekhar S, Gunturi V, Evans MR, Yang K (2012) Spatial big-data challenges intersecting mobility and cloud computing. In: Proceedings of the eleventh ACM international workshop on data engineering for wireless and mobile access. ACM, New York, pp 1–6

Strano E, Nicosia V, Latora V, Porta S, Barthélemy M (2012) Elementary processes governing the evolution of road networks. Sci Rep 2. Nature.com

Yang K, Gunturi VMV, Shekhar S (2012) A Dartboard Network Cut Based Approach to Evacuation Route Planning: A Summary of Results. In: Xiao N, Kwan MP, Goodchild MF, Shekhar S (eds) Geographic Information Science. GIScience 2012. Lecture Notes in Computer Science, vol 7478. Springer, Berlin/Heidelberg

Zikopoulos P, Eaton C (2011) Understanding big data: analytics for enterprise class hadoop and streaming data. McGraw-Hill Osborne Media

Spatial Joins

▸ Query Processing: Joins

Spatial Joins in Clusters

▸ Query Processing: Joins

Spatial Networks Big Data

▸ Spatial Graph Big Data

Spatio-social Data

Mohamed Sarwat and Yuhan Sun
School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ, USA

Synonyms

GeoSocial data; Spatial data

Definitions

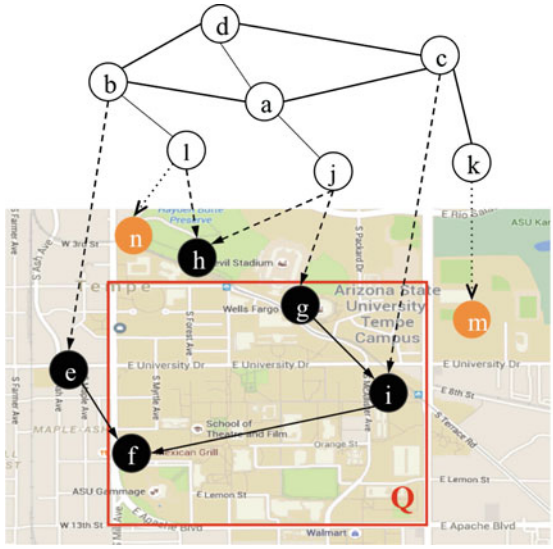
In the past decade, social networking services, e.g., Facebook and Twitter, managed to unprecedentedly connect hundreds of millions of people all over the world. Users register to online social networks in order to keep in touch with their friends and family, learn about their news, get recommendations from them, and engage in online social events. Thanks to the widespread use of mobile and wearable devices, popular social networks, e.g., Facebook, prompt users to add spatial attributes to social entities, e.g., check-ins, posts, and geo-tagged photos. Such spatial attributes are already being tied to social networking data to form what we call, *Spatio-Social Data*. Spatio-Social data brings both the physical space, social relationships, user interactions, and social media content into effect together, which led to the rise of many interesting applications. For instance, users in a location-based social networking service (e.g., Foursquare and Facebook Places) are associated with a geo-location and might alert friends when visiting a place (e.g., restaurant, bar) by checking in on their mobile phones.

Overview

Figure 1 gives an example of how Spatio-Social data looks like. The figure depicts a social graph in which there exist three types of entities: (i) *person* (e.g., Alice, Bob, Carol, Dan, etc.) with a name and age attributes, (ii) *place* (e.g., restaurants) with a name and a spatial location attributes, and (iii) *content* (e.g., tweet) with id, social media content (e.g., Tweet text), and location attributes. The graph consists of four types of social connections: (i) *friend of* that represents the social relationship between persons, e.g., Alice is a friend of Dan; (ii) *visited* that represents the users' visits to places, e.g., Dan visited Pita Jungle restaurant in Tempe AZ; (iii) *posted* that represents the relationship between people and social media content, e.g., Mat posted a tweet within the ASU stadium area; (iv) *rated* that represents the users' interactions with the physical space, e.g., Kate gave a five-star rating to Sushi 101.

Spatio-Social Graph

In a GeoSocial graph, a user may issue the following query: "Find Restaurants in Phoenix that are visited by Alice' s Friends." In addition, such data can be utilized to process GeoSocial analytic tasks, e.g., "Report restaurants in Phoenix that are visited by teenagers ($13 < \text{age} < 18$)." We call such queries that search the social graph with both social and spatial predicates, *GeoSocial Graph Search* (G^2S) *queries*. We can formally define a GeoSocial Graph Search G^2S query as a regular path query with geospatial predicates performed on the social graph. A G^2S query takes as input a sequence of social graph predicates as well as a spatial predicate. The social predicate is applied to traditional social graph entities and/or connections with nonspatial attributes, e.g., person, whereas the spatial predicate is performed on social entities that possess spatial attributes, e.g., restaurants. G^2S then returns a set of social graph paths that match the social predicates as well as satisfy the spatial predicate. Existing graph database systems, e.g., Neo4j, allow users to define spatial properties on graph elements. MongoDB is utilized by Armenatzoglou et al. (2013)



a: {name: Alice, age: 19}
b: {name: Dan, age: 20}
c: {name: Carol, age: 35}
d: {name: Bob, age: 25}
j: {place: Kate, age: 18, …}
k: {place: Mat, age: 23, …}
l: {place: Katharine, age:21, …}

e: {place: Pita Jungle, location:<x1,y1>}
f :{place: Chipotle, location:<x2,y2>}
g: {place: Sushi 101,…}
h: {place: Subway,…}
i: {place: McDonald's,…}

m: {id: 1, tweet: "Nice Weather", location:<x10,y10>}
n: {id: 2, tweet: "Good Game", location:<x10,y10>}

Spatio-social Data, Fig. 1 GeoSocial graph data

to manage both social and geographic data. Each user in the GeoSocial graph is assigned a spatial location. A Range Friends query searches for friends within a given distance from the user. The Nearest Friend query searches the nearest friend of a user.

Geo-tagged Social Media

Social media content is rich in content, e.g., text, video, hyperlinks, photos, and spatial location. Existing work studied keyword search on streaming social media data (Busch et al. 2012; Wu et al. 2013; Yao et al. 2012). Other newly emerging applications on GeoSocial media include event detection (Abdelhaq et al. 2013; Li et al. 2012; Marcus et al. 2011; Watanabe et al. 2011), news extraction (Sankaranarayanan et al. 2009), and analysis (Tweet Tracker 2013). Bao et al. (2012) propose a GeoSocial news feed, which ranks the news feed based on both the social and geospatial proximity of each post to the user.

User Check-In Data

Myriad Web applications produce *location-based ratings* that embed user and/or item locations. For example, applications like Foursquare and Yelp allow users to "check in" at spatial destinations (e.g., restaurants) and rate their visit and thus are capable of associating both user and item locations with ratings. Existing recommendation techniques assume ratings are represented by the (*user*, *rating*, *item*) triple and thus are ill-equipped to produce location-aware recommendations.

Recent systems produce *three* main types of location-based user interactions (Sarwat et al. 2014; Levandoski et al. 2012a): (1) Spatial ratings for nonspatial items, represented as a four-tuple (*user*, *ulocation*, *rating*, *item*), where *ulocation* represents a user location. Items are nonspatial in nature (e.g., movies) and thus do not have an associated location. As an example, traditional e-commerce applications (e.g., Netflix) may use a user's home address as *ulocation*, while mobile applications may associate the location where the user rated the item as the *ulocation*. (2) Nonspatial ratings for spatial items, represented as a four-tuple (*user*, *rating*, *item*, *ilocation*), where *ilocation* represents an item location. Examples of applications that produce such ratings are e-commerce applications that gather user opinions on venues/destinations (e.g., restaurant review websites), but do not collect user locations.

Key Research Findings

Spatio-Social Graph Queries

Basic approaches that answer GeoSocial Graph Search (G^2S) queries fall into two main categories, listed as follows:

(1) *Social-Then-Spatial (SoSpa)* approach This approach leverages the query processor of an existing graph data management system (Kyrola et al. 2012; Prabhakaran et al. 2012; Sarwat et al. 2012, 2013b; Shao et al. 2013). It first traverses the social graph to find matching paths (Chen and Chen 2008; Fan et al. 2011; Jin et al. 2010). When a spatial entity is encountered during the social graph traversal, SoSpa tests these entities against the spatial predicate. Finally, SoSpa returns only paths that satisfy both the social path predicates and the spatial range predicate.

(2) *Spatial-Then-Social (SpaSo)* approach This approach runs in two sequential phases:
   (a) Spatial filtering: This phase employs a state-of-the-art spatial database management system that harnesses a spatial index, e.g., R-tree, to first retrieve the social graph entities that satisfy the spatial predicate.
   (b) Social filtering: This phase traverses the social graph to only return social graph paths that match the social predicates and contains only the spatial entities retrieved in the spatial filtering phase. Even though both approaches correctly answer G^2S queries, nonetheless they may lead to performance penalties by traversing unnecessary graph paths that incur extra I/O and CPU overhead which increases the overall latency of G^2S queries (Sarwat and Sun 2017).

Spatio-Social Media Queries

Existing systems provide scalable indexing mechanisms that allow users to interactively explore spatio-textual data at scale. Various spatial and spatio-textual indexing techniques are explored in the literature (Budak et al. 2014; Magdy et al. 2014; Skovsgaard et al. 2014) to support spatial queries on geo-tagged social media. Novel systems extend those techniques to reduce the indexing latency and increase the digestion rate. Existing research work addresses the problem of tuning the in-memory part of the spatio-textual index to minimize the overall index storage and maintenance overhead. Initial studies show that space-partitioning spatial indexes achieve better performance in processing real-time social media, e.g., spatial grid index or spatial quad tree. On the contrary to data-partitioning spatial indexes, e.g., R-tree, cell boundaries in space-partitioning indexes do not change with the incoming data. Instead, each cell covers a specific area of the space and holds whatever data it contains. Recent research work also studies the problem of real-time geo-tagged social media content insertion in the spatio-textual, spatio-visual index by employing lazy batch updates so that the amortized insertion cost per social media content is minimized. A large body of work investigates compression techniques to minimize the memory footprint of geo-tagged social media to efficiently utilize main memory and storage resources. Such systems investigate storing only necessary information in main memory, e.g., recent tweets or local top frequent keywords, and get rid of any other information so that incoming spatial exploration queries can fetch its answers in main memory and avoid hitting the relatively slower secondary storage.

Spatio-Social Recommendation

Existing spatial database systems allow users to look up spatial data that matches exactly the designated query and hence do not leverage the user interaction with the physical space. Recommendation (Sarwat et al. 2017; Levandoski et al. 2012b; Herlocker et al. 2004; Koutrika et al. 2009) is the process of suggesting useful data to the user based on a large pool of historical user interaction data (e.g., a user rating a venue on Yelp). To take into account the user interaction with the physical space, existing approaches extend spatial data structures with user interaction data to support spatial data recommendation. The straightforward approach constructs a spatial

index, e.g., R-tree, on all spatial data objects and calculates the similarity between different items and data points. When the user asks for a set of spatial objects in a certain boundary region, the system first searches for all the points lying within the region. It is not feasible to go through all the data point outcomes from the search. For this purpose, the system filters huge data ignoring all those points that may not interest him based on his previous interactions with the system. Then, only the items with top-k recommendation scores are displayed to the user. The performance can be further optimized to give faster recommendations by augmenting existing spatial indexes to filter spatial data using both their spatial attributes and their recommendation scores.

The system produces recommendations by employing an adaptive hierarchical grid structure to partition data ratings by their *user location* attribute into spatial regions of varying sizes at different hierarchies. For a querying user located in a region R, we apply an existing collaborative filtering technique that utilizes only the ratings located in R. Existing systems can dynamically balance scalability and recommendation locality. In this case, the system produces recommendations by using *travel penalty*, a technique that penalizes recommendation candidates the further they are in travel distance to a querying user (Sarwat et al. 2013a). The challenge here is to avoid computing the travel distance (Hjaltason and Samet 1999) for all spatial items to produce the list of k recommendations, as this will greatly consume system resources. Existing systems address this challenge by employing an efficient query processing framework capable of terminating early.

Future Direction for Research

Spatio-Social data is rich in semantics. The graph model can carry structure information, spatial information, textual information, and user data at the same time. Such data is inherently heterogeneous; hence there is always room for novel applications that combine a variety of Spatio-Social aspects. Another direction of future research is to explore the appropriate approach to analyze and explore Spatio-Social data using visual map interfaces. The challenge is that Spatio-Social data is growing at a staggering rate. In that case, an interesting area of research will be to extend de facto distributed computing systems and streaming engines to support fast and scalable processing of Spatio-Social data.

Cross-References

- Linked Geospatial Data
- Spatial Graph Big Data

References

Abdelhaq H, Sengstock C, Gertz M (2013) EvenTweet: online localized event detection from Twitter. In: VLDB

Armenatzoglou N, Papadopoulos S, Papadias D (2013) A general framework for geo-social query processing. Proc VLDB Endow 6(10):913–924

Bao J, Mokbel MF, Chow C-Y (2012) Geofeed: a location aware news feed system. In: 2012 IEEE 28th international conference on data engineering. IEEE, pp 54–65

Budak C, Georgiou T, Agrawal D, Abbadi AE (2014) GeoScope: online detection of geo-correlated information trends in social networks. In: VLDB

Busch M, Gade K, Larson B, Lok P, Luckenbill S, Lin J (2012) Earlybird: real-time search at Twitter. In: ICDE

Chen Y, Chen Y (2008) An efficient algorithm for answering graph reachability queries. In: Proceedings of the IEEE international conference on data engineering, ICDE

Fan W, Li J, Ma S, Tang N, Wu Y (2011) Adding regular expressions to graph reachability and pattern queries. In: Proceedings of the IEEE international conference on data engineering, ICDE

Herlocker JL, Konstan JA, Terveen LG, Riedl JT (2004) Evaluating collaborative filtering recommender systems. ACM Trans Inf Syst TOIS 22(1):5–53

Hjaltason GR, Samet H (1999) Distance browsing in spatial databases. ACM TODS 24(2):265–318

Jin R, Hong H, Wang H, Ruan N, Xiang Y (2010) Computing label-constraint reachability in graph databases. In: Proceedings of the ACM international conference on management of data, SIGMOD

Koutrika G, Bercovitz B, Garcia-Molina H (2009) FlexRecs: expressing and combining flexible recommendations. In: Proceedings of the ACM international conference on management of data, SIGMOD, pp 745–758

Kyrola A, Blelloch G, Guestrin C (2012) PowerGraph: distributed graph-parallel computation on natural graphs. In: Proceedings of the USENIX symposium on operating system design and implementation, USENIX OSDI

Levandoski JJ, Sarwat M, Eldawy A, Mokbel MF (2012a) LARS: a location-aware recommender system. In: Proceedings of the international conference on data engineering, ICDE

Levandoski JJ, Sarwat M, Mokbel MF, Ekstrand MD (2012b) RecStore: an extensible and adaptive framework for online recommender queries inside the database engine. In: Proceedings of the international conference on extending database technology, EDBT

Li R, Lei KH, Khadiwala R, Chang KC-C (2012) TEDAS: a Twitter-based event detection and analysis system. In: ICDE

Magdy A, Mokbel MF, Elnikety S, Nath S, He Y (2014) Mercury: a memory-constrained spatio-temporal real-time search on microblogs. In: ICDE, pp 172–183

Marcus A, Bernstein MS, Badar O, Karger DR, Madden S, Miller RC (2011) Twitinfo: aggregating and visualizing microblogs for event exploration. In: CHI

Prabhakaran V, Wu M, Weng X, McSherry F, Zhou L, Haridasan M (2012) Managing large graphs on multi-cores with graph awareness. In: Proceedings of the USENIX symposium annual technical conference, USENIX ATC

Sankaranarayanan J, Samet H, Teitler BE, Lieberman MD, Sperling J (2009) TwitterStand: news in Tweets. In: SIGSPATIAL

Sarwat M, Sun Y (2017) Answering location-aware graph reachability queries on geosocial data. In: Proceedings of the international conference on data engineering, ICDE

Sarwat M, Elnikety S, He Y, Kliot G (2012) Horton: online query execution engine for large distributed graphs. In: Proceedings of the international conference on data engineering, ICDE

Sarwat M, Eldawy A, Mokbel MF, Riedl J (2013a) PLUTUS: leveraging location-based social networks to recommend potential customers to venues. In: Proceedings of the international conference on mobile data management, MDM

Sarwat M, Elnikety S, He Y, Mokbel MF (2013b) Horton+: a distributed system for processing declarative reachability queries over partitioned graphs. PVLDB 6(14):1918–1929

Sarwat M, Levandoski JJ, Eldawy A, Mokbel MF (2014) LARS*: an efficient and scalable location-aware recommender system. IEEE Trans Knowl Data Eng TKDE 26(6):1384–1399

Sarwat M, Moraffah R, Mokbel MF, Avery JL (2017) Database system support for personalized recommendation applications. In: Proceedings of the international conference on data engineering, ICDE

Shao B, Wang H, Li Y (2013) Trinity: a distributed graph engine on a memory cloud. In: Proceedings of the ACM international conference on management of data, SIGMOD

Skovsgaard A, Sidlauskas D, Jensen CS (2014) Scalable top-k spatio-temporal term querying. In: ICDE, pp 148–159

Tweet Tracker (2013) TweetTracker: track, analyze, and understand activity on Twitter. http://tweettracker.fulton.asu.edu/

Watanabe K, Ochi M, Okabe M, Onai R (2011) Jasmine: a real-time local-event detection system based on geolocation information propagated to microblogs. In: CIKM

Wu L, Lin W, Xiao X, Xu Y (2013) LSII: an indexing structure for exact real-time search on microblogs. In: ICDE

Yao J, Cui B, Xue Z, Liu Q (2012) Provenance-based indexing support in micro-blog platforms. In: ICDE

Spatiotemporal Data Integration

▸ Spatial Data Integration

Spatiotemporal Data: Trajectories

Xiaofang Zhou and Lei Li
School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, QLD, Australia

Synonyms

Moving objects; Routes; Spatiotemporal representation; Traces

Definitions

Let $p(l, t)$ be a *spatiotemporal point* with location l at time t. A *trajectory* is defined as $\tau = < p_1, p_2 \ldots p_n >$ where $p_i.t \leq p_j.t$ if $i < j$. That is, a trajectory is a sequence of spatiotemporal points ordered by time.

Location l can be represented as a longitude and latitude pair in geographical space or a road segment ID and distance offset in a road network. A trajectory without temporal information is often called *route* or *path*, and a collection of

trajectories of an object is called its *trace*. The trajectory with a specific origin and destination pair (OD pair) is also called a *trip*.

Overview

A trajectory records how an object moved in a space. Such information is easier than ever to acquire with the prevalence of location-capturing devices such as GPS nowadays. Therefore, large volumes of trajectory data are being accumulated from various sources every day, for animals, human, vehicles, and natural phenomena (Zheng 2015). Animal trajectory data can be obtained by attaching tracking devices to animals, for environment protection and animal behavior studies. *Movebank* has collected animal movement data from thousands of studies at millions of locations. Human trajectories are collected from travelers, cyclists, and joggers, due to the recent popularity of electronic fitness tracking devices and mobile devices. Transport-related trajectory data, which by far are the most voluminous, most interesting, and most useful type of trajectory data, are generated by GPS devices and fixed-location data-capturing devices from vehicles, airplanes, and ships. Taxi service providers like *Uber* and *DiDi* create terabytes of trajectory data every single day. Natural phenomena trajectory data are also collected for scientific studies. *NOAA Air Resources Laboratory* (Draxler and Rolph 2003) stores a massive amount of meteorology trajectories that can be used to better understand the causes and impacts of natural disasters and to protect the natural environment.

The works on trajectory data can be categorized into several topics (Zheng and Zhou 2011). The first one is *trajectory preprocessing*, including noise removal to improve data quality, map matching that aligns points to road segments for road network-constrained moving objects (such as cars), data compression, and trip segmentation that prepares data for further uses like clustering and classification. The second one is related to *trajectory data management*, which aims at answering retrieval queries efficiently by building indexes and developing query algorithms. The third one is *trajectory mining*, which involves finding the patterns among the trajectories, classifying them into different categories, detecting outliers, and reducing the uncertainty between two consecutive points. Finally, based on all the previous steps, trajectory data can be used to solve problems ranging from more conventional applications such as traffic condition prediction and route planning to the more recent applications such as fuel and pollution emission minimization in a city.

Key Research Findings

Trajectory Preprocessing

Essentially, the raw trajectory is an array of (l, t) data, which can be noisy, too dense, or too coarse in terms of sampling rates and cannot be directly used for a variety of applications. Therefore, like any other types of raw data, preprocessing is needed before actual uses.

Noise Removal

Due the accuracy of the devices, the data collected are not always accurate. Some of the data points obviously drift off the course. The simplest approach is using the mean or median value of a sliding window to filter out the noise point. However, it fails when there are multiple consecutive noise points. More practical approaches apply outlier detection methods, like computing the travel speeds of the points and removing those that surpass the threshold (Yuan et al. 2013).

Map Matching

If there exists an underlying road network that confines object movement (e.g., for cars), it is always beneficial to attach the GPS points to the corresponding roads. Based on the time when the matching is executed, it can be categorized into *real-time mode* and *post-processing mode*. The real-time map matching is widely applied in real-time turn-by-turn navigation systems. It requires fast computation and can only use the previous few points (i.e., no future points can be used), while it cannot guarantee the continuity on the path during the trip. The post-processing map

matching can utilize the entire trajectory, so it is more accurate but time-consuming.

The techniques used in map-matching methods can be divided into four groups. The first one mainly considers the geometry distance between GPS point/trajectory segment and the candidate map points/map edges that could be possibly aligned on. The second group also considers topology of a map such as connectivity and contiguity of roads. The third group further improves the accuracy by using probability-based methods like the *Hidden Markov Model* and *Kalman Filter*. More advanced approaches combine the existing methods with additional information like Wi-Fi, Bluetooth, and cellular fingerprint on mobile phones, driver behaviors, and other semantic information about the road network, the objects, and other related information.

Compression

The amount of trajectory data increases at an increasing pace, leading to gigantic storage overhead as well as computation and communication costs. However, not many applications need the trajectory data to be that precise, so compression is necessary in many cases.

A simple approach to reduce the size is to remove some points if they do not affect the precision dramatically. In this way, the new and more compact trajectory is an approximation of the original one. Another approach takes advantages of the road network if applicable. Data size can be reduced significantly after using consecutive matched road segments to represent points because normally there are multiple GPS points along the same road segment. Further compression can be achieved with the help of frequent sequential pattern mining and Huffman Coding (Song et al. 2014), or using other string compression methods like *Burrows-Wheeler Transform*, because a series of roads can be viewed as a string with each road representing a character in the alphabet (Koide et al. 2017).

Segmentation

In many high-level applications where trajectory data are used (such as traffic and traveler behavior analytics), shorter trajectory segments make more sense than the original long trajectory. This is not only because shorter trajectories can better support similarity-based analysis but also improve computation efficiency (many trajectory similarity measures are of quadratic complexity). Further, segmentation based on OD pairs can also bring semantic information to trajectories. Trajectory segmentation (or trip segmentation) is the process that breaks a long trajectory into a series of short trajectories.

The segmentation processing has three main categories. Firstly, the trajectory can be segmented based on time interval. It is like resampling the original trajectory on a lower sampling rate. Secondly, it can be segmented based on the shape (Lee et al. 2007), which involves finding the turning points. Finally, semantic meaning of the points (like walk segment, driving segment, and segments between taxi waiting time) can also serve as segmentation points.

Trajectory Management

Querying and processing directly on a large volume of trajectories are actually very time-consuming. Therefore, how to organize and index the trajectory data to support trajectory query answering efficiently becomes a research topic, which is called trajectory management (Deng et al. 2011).

Trajectory Query

Based on the type of query entity (i.e., points, regions, and trajectories), trajectory queries can be classified into three types. *P-Query* asks for points which satisfy a given spatiotemporal relationship to specified trajectory segment(s) (e.g., top-k nearest neighbors) or reversely. Similarly, *R-Query* asks for regions, and *T-Query* asks for trajectories. An example of a spatiotemporal relationship is "within 500 m of a gas station between 9:00pm and 9:30pm."

Trajectory Index

Compared with other general data types, trajectory data has unique characteristics like continuous long time span. Meanwhile, queries on trajectory also often ask for information

in a continuous time window. Based on these characteristics, three types of indexes are proposed.

The first type augments the existing multi-dimensional index with a temporal dimension, like *3D R-Tree*. The second type further breaks the temporal dimension down to multi-version structures, such as *HR^+-Tree* and *MV3R-Tree* (Tao and Papadias 2001). The third type focuses on dividing the spatial dimension into grids and then building a separate temporal index on each grid. This category includes *SETI* and *MTSB-Tree*.

Trajectory Similarity

Like any other data types, a similarity (or distance) measurement is needed to compare between trajectories.

The simplest scenario is the distance from a point to a trajectory, which is measured by the distance to the nearest point in the trajectory. As for the distance between a set of nodes and a trajectory, the closer matched pair of points is assigned with larger weights using the sum of distance, while those faraway pairs are given much lower value typically in an exponential way.

The similarity between two trajectories is usually measured by some kind of aggregation of distances between trajectory points (Wang et al. 2013). Along this line, several typical similarity functions for different applications include *Closest Pair Distance*, *Sum-of-Pairs Distance*, *Dynamic Time Warping*, *Longest Common Subsequence*, *Edit Distance with Real Penalty*, and *Edit Distance on Real Sequences*. It is worth noting that some of those similarity functions were originally proposed for time series data. But as trajectories can be regarded as a special kind of time series in a multidimensional space, these similarity functions can also be applied to trajectory data.

Trajectory Uncertainty

Because trajectory data is always a sample of the object's actual movement trace, the uncertainty exists between any two points in a trajectory especially when the sampling rate is low (Zheng et al. 2012). On one hand, some works aim to reduce the uncertainty of the trajectory. On the other hand, other works try to add more uncertainties for privacy protection reasons.

The first group of researches focuses on providing conservative bounds for the positions of uncertain objects between two points, which is achieved by employing geometric cylinders or beads. Independent probability density functions can be used to model the uncertain positions (Cheng et al. 2004).

The other group of approaches aim at providing the most k likely routes between sample points with the help of a set of uncertain trajectories, because these trajectories that share similar routes can often supplement each other to make themselves more complete (Su et al. 2013).

Contrary to the previous attempts, techniques are developed to work on preventing user privacy leaking by blurring the published trajectory while preserving the utility of the data.

Trajectory Mining

Trajectory Pattern Mining

Trajectory pattern mining aims at discovering trajectory groups based on their proximity in either a spatial or a spatiotemporal sense. There are four main categories of patterns that can be discovered from a single trajectory or a group of trajectories.

The first one is the *moving together pattern*, which discovers a group of objects that move together for a certain time period. A *flock* is a group of objects that travel together within a disk of some user-specified size for at least k consecutive timestamps. Apparently, the fixed disk shape with a fixed size can be too strict and rigid to use in practice, so *convoy* is proposed by finding patterns based on density. In this way, patterns of any shape and size can be discovered. However, both flocks and convoys are strict on period, so *swarm* (Li et al. 2010a) is proposed to further generalize the cluster with objects lasting for at least k timestamps. To cope with stream data, *traveling companion* uses a data structure (called traveling buddy) to continuously find convoy-/swarm-like patterns from trajectories and can work online.

By allowing the membership of a group to evolve gradually, *gathering* (Zheng et al. 2014) can be used to detect events and incidents.

The second one is *trajectory clustering*. Unlike general clustering tasks that use feature vectors to represent objects, it is hard to generate a uniform feature vector because different trajectories contain different and complex properties, such as length, shape, sampling rate, number of points, and their orders. A number of works have been done using the trajectory similarity. Although some of them work on the entire trajectory, it is rare for two objects traveling together for the entire journey. So more practical approaches partition trajectories into segments before clustering. If the trajectories are matched to map, the trajectory clustering task can be done by applying graph clustering algorithms.

The third one is *mining sequential patterns from trajectories*. A *sequential pattern* means a certain number of moving objects travel a common sequence of locations in a similar time interval and the locations of the sequence do not have to be consecutive. A general solution is using trajectory clustering first and then reforming trajectories with cluster IDs. In this way, existing sequential pattern mining algorithms like *PrefixSpan* can be used. If the trajectory can be matched on map, the resulting sequence of road IDs can use *Suffix Tree* to find the frequent patterns (Song et al. 2014).

The last one is *mining periodical patterns from trajectories*. Some object movements have periodical patterns over the long history. For example, people go to work in the morning and go back home at night. Animals migrate from one place to another at different time of the year. A straightforward approach is to use general frequent pattern mining methods. However, real-life periodic behaviors are complicated and involve multiple interleaving periods, partial time span, and spatiotemporal noises and outliers. Therefore, a more advanced two-stage method is proposed (Li et al. 2010b). In the first stage, it mines all the frequent visiting places by density-based clustering algorithms. The temporal data corresponding to entering and leaving these places can be used to find the period values. In the second stage, larger periodic patterns are created by applying hierarchical clustering algorithms on the partial movement sequences.

Trajectory Classification

Trajectory classification helps divide trajectories into different statuses. For example, taxi trajectories can be *occupied, non-occupied*, and *parking*. A cell phone user can be *stationary, walking*, and *driving* or even *driving, biking, commuting by bus*, and *walking*. In general, the classification has three steps. First of all, trajectories are divided into segments in preprocessing stage. After that, features of each segment are extracted. Finally, existing sequence inference models (such as *Dynamic Bayesian Network*, *Hidden Markov Model*, and *Conditional Random Field*) can be used.

Outliers Detection

Outliers of trajectory data can be points significantly different from others spatially or temporally and can also be observations that do not follow the expected patterns or constraints. One general approach is to leverage standard frequent pattern mining methods. If the trajectory cannot fall into any cluster, it might be an outlier (Lee et al. 2007).

Examples of Application

Trajectory data can be found in many applications; here we just list a few as examples.

Travel Recommendation. It aims to find interesting locations and travel sequences from trajectories generated by many people. (Zheng and Xie 2011) identifies staying points from users' trajectories and clusters these points into locations of interest. After that, it can identify the top-k most interesting locations and travel experts in a city and do the recommendations based on their data. Moreover, it can recommend trajectories themselves, because historical traveling experiences can also reveal valuable information on how other people usually choose routes between locations.

Traffic Condition Estimation. The trajectories of vehicles on the road can reflect the traffic condition (Yang et al. 2013). It needs a series of

processing to generate a speed profile from trajectories: map matching, speed generation, missing value estimation, and compression. The result not only can be used for finding the fastest path from one place to another at different departure time but also can help find the congestion of road network and provide decision support for urban planning.

Map Inference. Normally, vehicle trajectories can always be matched to some roads on a map. However, when a new road is developed and the map has not been updated, or if there is even no map for the current region, map matching will fail. Map inference works under this scenario to infer new maps or update existing maps based on trajectories.

Diagnosing Traffic Anomalies. Such examples can be that a taxi driver takes a malicious detour, that an unexpected road change occurs, or that people travel a wrong path. They can all be discovered by trajectory outlier detection using trajectory clustering.

Future Movement Prediction. Periodic trajectory pattern mining can be used to predict the next direction or destination of the current moving objects, like a group of animals or a commuter. The prediction can further help compress the trajectory data itself.

User Similarity Estimation. There are many aspects of similarity between users, like connections from social network, point of interest, check-in history, and any other logs, that can be obtained. Besides them, trajectory data, which reveals the life patterns of the users by trajectory classification and clustering, can also help improve the accuracy of similarity comparison.

Sport Tactic Analysis. For many team games like soccer, basketball, hockey, and rugby, the players' movements are essentially trajectories. By analyzing the video data from different cameras, the trajectory of each player can be reconstructed and are used in tactic analysis by many professional clubs nowadays. Some of them even hire a data analyst as a coaching staff.

Airspace Monitoring and Aircraft Guiding. A large volume of aircraft trajectory data is managed by an air traffic controller. They use these trajectories to monitor the "health" of the airspace, which is implemented by trajectory clustering and outlier detection.

Scientific Study. Meteorologists use trajectory of SO_2 and NO_x in an isentropic and constant level to analyze the contributions of acidic deposition. Zoologists use the trajectory of animals to study their movement patterns. Biologists use proteomic trajectories to study mouse retina development.

Future Directions for Research

As summarized above, there exists a rich body of research on all aspects of trajectories, from data capturing, cleaning, compression, and indexing to processing. We have witnessed an accelerating trend of research activities form both academic and industry on this topic. There are three main drivers in today's big data landscape, which will also drive the research in trajectory data management, processing, and analytics in the foreseeable future.

First, it is about volume. Not only the volume of trajectory data now can reach TB level daily for some large navigation or taxi-/car-sharing companies, but also the number of queries has increased dramatically. What is the best way to process map matching for one billion points every day? How can we reduce the processing costs if we have 10,000 shortest path queries in the same region? This is a scenario that exists already, as when a user opens a map-based app and inputs a location, it issues a shortest path query. For every problem we solved before, it is the time to revisit them, to see how they can become scalable using new computing platforms and how better algorithms can be designed to support batch query processing (for a very large number of streaming queries on streaming data).

Second, it is about semantics. After all, trajectory data are low-level data which can be noisy and with a high level of redundancy (among consecutive points of one moving object, among the history data of one moving object over a long period, and among objects with similar moving patterns). There is a dilemma between justifying

the high costs of storing all data available and the fear that some data which we only find useful in the future for some purposes we do not know yet might be lost if we do not store them. Clearly, a new way of thinking is needed to manage trajectory data based on a semantic hierarchy, from raw data, calibrated data, events detected, summarization of data for a basic set of requirements, patterns discovered, and general statistics. Data can be gradually reduced over time and eventually removed. In such a way, trajectory data can be at the center for data integration and data analytics, as location and time are two ubiquitous dimensions for most information useful to us. Trajectory will no longer be considered as specialized data with limited applications; rather, it is an enabler data asset underpinning the future of a data-rich society.

Third, trajectory data can reveal so much about a person. With so much location and movement data about a person captured and accessed so easily, security and privacy become an extremely serious issue for trajectory data. Simple facts about one visiting or not visiting a place can be highly sensitive. This problem can become much more significant when the trajectories are used with other data sources including social network data. Some research has already started on this topic, but much more is needed urgently.

Cross-References

- Big Data and Privacy Issues for Connected Vehicles in Intelligent Transportation Systems
- Big Data in Smart Cities
- Data Cleaning
- Indexing
- Spatial Data Mining
- Storage Technologies for Big Data

References

Cheng R, Kalashnikov, DV, Prabhakar, S (2004) Querying imprecise data in moving object environments. IEEE Trans Knowl Data Eng 16(9):1112–1127

Deng K, Xie K, Zheng K, Zhou X (2011) Trajectory indexing and retrieval. Computing with spatial trajectories. Springer, New York, pp 35–60

Draxler RR, Rolph, GD (2003) Hysplit (hybrid single-particle lagrangian integrated trajectory). NOAA air resources laboratory, silver spring, MD. model access via NOAA ARL ready website

Koide S, Tadokoro Y, Xiao C, Ishikawa Y (2017) CiNCT: compression and retrieval for massive vehicular trajectories via relative movement labeling. arXiv preprint arXiv:1706.02885

Lee J-G, Han J, Whang K-Y (2007) Trajectory clustering: a partition-and-group framework. In: Proceedings of the 2007 ACM SIGMOD international conference on Management of data. ACM, pp 593–604

Li Z, Ding B, Han J, Kays R (2010a) Swarm: mining relaxed temporal moving object clusters. Proc VLDB Endow 3(1–2):723–734

Li Z, Ding B, Han J, Kays R, Nye P (2010b) Mining periodic behaviors for moving objects. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 1099–1108

Song R, Sun W, Zheng B, Zheng Y (2014) Press: a novel framework of trajectory compression in road networks. Proc VLDB Endow 7(9):661–672

Su H, Zheng K, Wang H, Huang J, Zhou X (2013) Calibrating trajectory data for similarity-based analysis. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data. ACM, pp 833–844

Tao Y, Papadias D (2001) The mv3r-tree: a spatio-temporal access method for timestamp and interval queries. In: Proceedings of very large data bases conference (VLDB), 11–14 Sept, Rome

Wang H, Su H, Zheng K, Sadiq S, Zhou X (2013) An effectiveness study on trajectory similarity measures. In: Proceedings of the twenty-fourth Australasian database conference, vol 137. Australian Computer Society, Inc., pp 13–22

Yang B, Guo C, Jensen CS (2013) Travel cost inference from sparse, spatio temporally correlated time series using Markov models. Proc VLDB Endow 6(9): 769–780

Yuan J, Zheng Y, Xie X, Sun G (2013) T-drive: enhancing driving directions with taxi drivers' intelligence. IEEE Trans Knowl Data Eng 25(1):220–232

Zheng Y (2015) Trajectory data mining: an overview. ACM Trans Intell Syst Technol (TIST) 6(3):29

Zheng Y, Xie X (2011) Learning travel recommendations from user-generated GPS traces. ACM Trans Intell Syst Technol (TIST) 2(1):2

Zheng Y, Zhou X (2011) Computing with spatial trajectories. Springer Science & Business Media, New York

Zheng K, Zheng Y, Xie X, Zhou X (2012) Reducing uncertainty of low-sampling-rate trajectories. In: 2012 IEEE 28th international conference on data engineering (ICDE). IEEE, pp 1144–1155

Zheng K, Zheng Y, Yuan NJ, Shang S, Zhou X (2014) Online discovery of gathering patterns over trajectories. IEEE Trans Knowl Data Eng 26(8):1974–1988

Spatiotemporal Graphs Big Data

▸ Spatial Graph Big Data

Spatiotemporal Networks Big Data

▸ Spatial Graph Big Data

Spatiotemporal Representation

▸ Spatiotemporal Data: Trajectories

Spatio-textual Data

Gao Cong[1] and Christian S. Jensen[2]
[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore
[2]Department of Computer Science, Aalborg University, Aalborg, Denmark

Synonyms

Geo-textual data; Spatial and textual data

Definitions

With the proliferation of GPS-equipped mobile devices, notably smartphones, massive volumes of geo-located, or geo-tagged, text content are becoming available. For example, Foursquare hosts over 105 million locations around the world with over 12 billion check-ins (https://foursquare.com/ about accessed January 2018), where each location is associated with both a geographical location and text content and similarly each check-in is also associated with a location and text. Facebook also supports location check-ins. We refer to such data as *geo-textual*, or *spatio-textual*, data. Other examples of such data include points of interest (POIs) with descriptive text, geo-tagged microblog posts (e.g., tweets), geo-tagged photos with text tags (e.g., as found at Flickr and Instagram), geo-tagged news, and geo-tagged web pages. Spatio-textual data can be divided into (i) streaming spatio-textual data that arrives at a high rate, exemplified by geo-tagged tweets, and (ii) static spatio-textual data that is relatively stable, exemplified by collections of POIs.

Overview

Massive volumes of spatio-textual data are available from many sources. Furthermore, as new spatio-textual data is being generated, the volumes will continue to grow at rapid pace as mobile devices continue to proliferate. Additionally, spatio-textual data often comes with rich context information, such as temporal information. With this development as the backdrop, many research problems and solutions have been proposed for managing, analyzing, and mining spatio-textual data. We proceed to cover three representative types of research problems.

Spatial keyword queries To support the querying or search of spatio-textual data, many types of spatial keyword queries have been proposed. In general terms, a spatial keyword query takes a location and keywords and arguments and finds the spatio-textual objects that best match the location and keywords. For example, a user may want to find a nearby place whose textual description is relevant to "fine arts." A number of index structures and query processing algorithms (Zhou et al., 2005; Hariharan et al., 2007; Felipe et al., 2008; Cong et al., 2009, 2012; Wu et al., 2012a, b; Rocha-Junior et al., 2011; Khodaei et al., 2010; Vaid and Jones, 2005; Zhang et al., 2013a, b; Chen et al., 2006; Christoforaki et al., 2011) have been developed for efficiently processing different kinds of spatial keyword queries.

Querying spatio-textual streams Spatio-textual data may arrive at a high rate (e.g., geo-tagged tweets, photos, or check-ins) and can then be modeled as data streams. In such streaming settings, continuous queries are of particular interest as users may want to be notified when interesting spatio-textual objects arrive. Several solutions (Chen et al., 2013, 2015; Li et al., 2013; Wang et al., 2015; Hu et al., 2015) for continuously querying spatio-textual streams have been proposed. For example, a user may want to be notified when tweets arrive that contain the term "flu" and are posted from within 5 km of the user's home.

Exploring spatio-textual data When users do not have clear ideas about what to query or search for, it is beneficial to provide functionality that enables users to search, navigate, and discover new facts from spatio-textual data. Several proposals exist on exploring spatio-textual data (Feng et al., 2016; Skovsgaard et al., 2014; Zhao et al., 2016). For example, when a use r is exploring a region on a map, the user can be shown the top-k most frequent terms in the region, and the result can be updated interactively when the user slides over the map.

Key Research Findings

Spatial keyword queries Consider a set $\mathcal{D}$ of spatio-textual objects. An object $p \in \mathcal{D}$ is a two-tuple: $\langle \lambda, \psi \rangle$, where λ encodes a geo-location and ψ is a text value. Many types of spatial keyword queries on spatio-textual data exist that target different applications. The most standard spatial keyword queries generalize fundamental queries from spatial databases and information retrieval. In spatial databases, the arguably most fundamental queries are range and k nearest neighbor queries. In information retrieval, queries may be Boolean keyword expressions, returning results that satisfy the Boolean expressions, or ranking-based, returning the k objects that are most similar to query keywords according to some text similarity function. Basic spatial keyword queries return a set or ranked list of objects from $\mathcal{D}$.

Four types of basic queries are covered below. Let ρ be a spatial region, λ be a point location, τ be a Boolean keyword expression, ψ be a set of keywords, and k be the number of objects to return. (1) A *Boolean range query* $q = \langle \rho, \tau \rangle$ returns all objects in $\mathcal{D}$ that are located in region ρ and that satisfy the Boolean expression τ. An example is "Retrieve all objects whose text description contains the keywords $vegetarian$, $pizza$, and $latte$ and whose location is within 1 km of the query location." (2) A *Boolean* kNN *query* $q = \langle \lambda, \tau, k \rangle$ returns up to k objects from $\mathcal{D}$, each of which satisfies the Boolean expression τ, ranked in increasing spatial distance from λ. An example is "Retrieve the k objects nearest to the query location such that each object's text description contains the keywords $vegetarian$, $pizza$, and $latte$." (3) A *top-k range query* $q = \langle \rho, \psi, k \rangle$ returns up to k objects from $\mathcal{D}$ that are located in the query region ρ, now ranked according to their text relevance to the set of keywords ψ. Here, an example is "Retrieve up to k objects whose locations are within 1 km of the query location, and that have the highest ranking scores, measured by the relevance of their text descriptions to the query keywords $vegetarian$, $pizza$, and $latte$." (4) A *top-k kNN query* $q = \langle \lambda, \psi, k \rangle$ retrieves k objects from $\mathcal{D}$, ranked according to a function that takes into consideration both spatial proximity and text relevance. This functionality is exemplified by the query "Retrieve the k objects with the highest ranking scores according to a function that combines their distance to the query location (a point) and the relevance of their text description to the query keywords $vegetarian$, $pizza$, and $latte$."

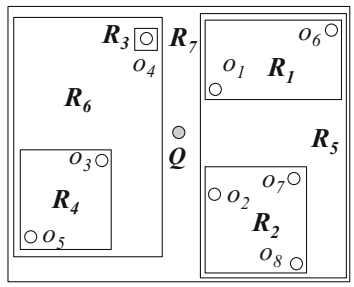
Index structures and algorithms have been developed to enable efficient processing of spatial keyword queries. Existing spatial keyword indexing techniques often combine a spatial index and a text index so that both textual and spatial information can be utilized to prune the search space when processing spatial keyword queries. The existing indices can be categorized according

to the spatial index they utilize: (i) R-tree-based indices (Zhou et al., 2005; Hariharan et al., 2007; Felipe et al., 2008; Cong et al., 2009; Wu et al., 2012a, b; Rocha-Junior et al., 2011), (ii) grid- or quad-tree-based indices (Khodaei et al., 2010; Vaid and Jones, 2005; Cong et al., 2012; Zhang et al., 2013a, b), and (iii) space-filling curve-based indices (Chen et al., 2006; Christoforaki et al., 2011).

The grid-based indices combine a spatial grid index and an inverted index for handling spatio-textual data, with two obvious ways of combining the two types of indices being the spatial primary index and the text primary index (Vaid and Jones, 2005). In the spatial primary index, spatio-textual objects are first organized by a grid index based on their spatial information. Then, for each grid cell, an inverted file is built for the objects that fall in the cell. In contrast, the text primary index extends the inverted index with the spatial information. Specifically, in the text primary index, each word is associated with a postings list, in which postings, each representing id of an object containing the word, are organized by a grid cell based on their locations. Each posting in a postings list contains the id of an object that contains the word of the postings list, and the postings are organized in a grid according to the locations of their objects. Using the example set of spatio-textual objects shown in Fig. 1a, b, the structure of the text primary index is exemplified in Fig. 2. The two index structures are proposed for tackling the problem of retrieving web documents relevant to a keyword query within a prespecified spatial region, i.e., the Boolean range query. For example, using the spatial primary index, at query time a set of cells that intersects with the query region is retrieved first. Then, the objects in these cells whose documents satisfy the query keywords are returned as the result. There exists no method to extend the two indices to efficiently handle the other three types of basic queries. A straightforward extension would not work better than using only an inverted file.

The R-tree-based geo-textual indices typically combine the R-tree index and the inverted index. The IR-tree (Cong et al., 2009) is a representative structure in this category. The index augments each node of the R-tree with a summary of the text content of the objects under the node. Specifically, each node contains a pointer to an inverted file that summarizes the text content of the objects in the subtree rooted at the node. The inverted file for a node contains (1) a vocabulary of all distinct terms in the text descriptions of the objects in the node's subtree and (2) each term t that is associated with a postings list, which is a sequence of pairs $\langle c, wt_{cp,t} \rangle$. Here, if the node is a non-leaf node, c is a child node of the node, and $wt_{c,t}$ is the number of times term t occurs in the object in the subtree that contains t the most times. If the node is a leaf node, c is an object in the node, and $wt_{c,t}$ is the frequency of

a



Spatio-textual objects and their bounding rectangles

b

object	terms and term frequencies
o_1	(Italian, 2)(restaurant, 2)(pasta, 1)
o_2	(coffee, 3)(restaurant, 3)(tasty, 1)
o_3	(Italian, 1)(pizza,1)(tasty, 1)
o_4	(restaurant, 1)(pizza, 1)(tasty, 1)
o_5	(coffee, 4)(restaurant, 3)
o_6	(Italian, 4)(restaurant, 4)
o_7	(Italian, 1)(coffee, 1)(restaurant, 4)(pasta, 1)
o_8	(coffee, 3)(restaurant, 3)(tasty, 1)

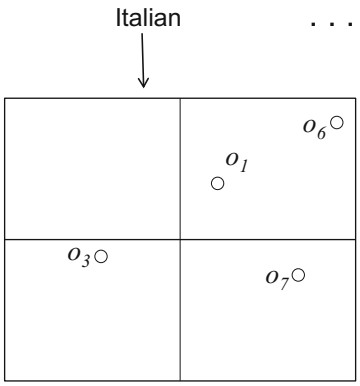
Text information of objects in (a)

Spatio-textual Data, Fig. 1 Example. (**a**) Spatio-textual objects and their bounding rectangles. (**b**) Text information of objects in (**a**))

term t in the object. The structure of an IR-tree is illustrated in Fig. 3.

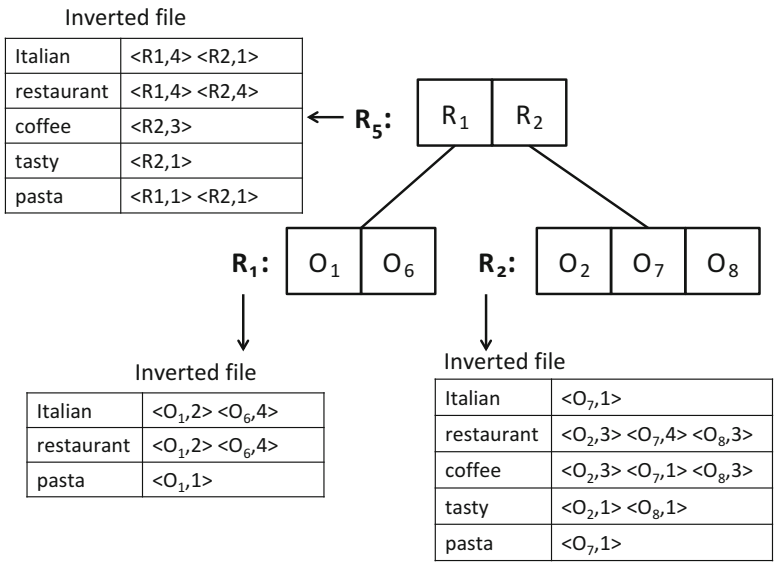
The IR-tree supports all the four types of basic queries. We proceed to use the Boolean range query and top-k kNN query to illustrate how the IR-tree can be used to handle the basic queries. The high-level algorithm for using the IR-tree to handle the Boolean range query starts at the root node of an IR-tree and checks each entry in the node. If an entry overlaps the query region and satisfies the keyword expression, the algorithm will check the entry's child node. The checking is done recursively over the IR-tree. When a leaf node is reached, the spatio-textual objects in the leaf nodes are checked, and objects satisfying both the spatial condition and the Boolean keyword condition are returned as results. Note that each non-leaf node contains summary information on both the spatial location and the terms of the objects in the node's subtree. For example, in node R_5 in Fig. 3 tells which terms the inverted file associated with the node will tell what keywords are contained in the objects in the node's subtree. If a query targets objects containing the term "rice," the inverted file tells that there is no need to search R_5's subtree.

The key idea of employing the IR-tree to compute the top-k kNN query is that each tree node on the summary information for each entry can be used to estimate a good upper bound on the relevance score, considering both spatial proximity and keyword relevance, to the query. This makes it possible to extend the best-first traversal algorithm, a standard algorithm for efficiently computing spatial kNN queries using the R-tree, to compute the top-k kNN query. The algorithm



Spatio-textual Data, Fig. 2 Grid structure for keyword *Italian*

Spatio-textual Data, Fig. 3 IR-tree and its inverted files



S

starts at the root and maintains a priority queue to keep track of the nodes and objects that have yet to be visited, for which the estimated upper bounds for each node, or the spatio-textual relevance score for each object, are used as the keys of nodes or objects, respectively. When deciding which node to visit next, the algorithm picks the node with the best key value from the set of nodes that have yet to be visited. The algorithm stops when the upper bounds of the remaining nodes are smaller than the currently k best results. The benefit of using the IR-tree for computing spatial keyword queries is that both spatial and textual information can be used in a combined fashion to prune the search space during query processing.

There also exist spatio-textual indices that combine a space-filling curve, e.g., the Z-order curve, and the inverted file. A typical example is the SFC-QUAD index (Christoforaki et al., 2011), which extends the inverted file with spatial information using space-filling curve. Specifically, the docIDs in each inverted list are assigned and ordered based on their spatial positions along a Z-order curve. SFC-QUAD is designed for *Boolean range query*. When processing a query with a given spatial region, a set of object ID ranges that fall in the query region are found. Then, these ranges are merged into a smaller number of ranges for reducing random disk I/O costs. Subsequently, the corresponding parts of the inverted lists of the query keywords within the ID ranges are retrieved, and then existing techniques for checking the Boolean expression of query keywords can be employed on the retrieved inverted lists. There exists no method to extend this type of index to efficiently handle the other three types of basic queries.

Beyond the basic spatio-textual queries, many other types of spatio-textual queries exist that are designed for different types of user needs. For example, the *m-closest keywords* (mCK) query (Zhang et al., 2009, 2010; Guo et al., 2015) retrieves a set of objects whose text content together contains a set of m query keywords and such that the maximum distance between any two objects is minimized. Another example is the *collective keyword* query (Cao et al., 2011, 2015; Long et al., 2013), which takes a location λ and a set of keywords ψ as arguments. Its search space is all subsets of the set of objects $\mathcal{D}$, and it returns a set of objects such that (i) the textual descriptions of these objects collectively cover ψ, (ii) the result objects are all close to λ, and (iii) the result objects are close to each other. These problems of retrieving a group of objects satisfying keyword covering condition while minimizing the distance within the group or between the group and query are NP-hard, and heuristic and approximation solutions are developed to answer these queries efficiently.

Querying streaming spatio-textual data On streaming spatio-textual data, the four kinds of spatial keyword queries can be posed as snapshot queries. Furthermore, subscription queries on streaming spatio-textual data are often of great interest. A typical type of subscription query on spatio-textual data streams is the *Boolean subscription query* (Chen et al., 2013; Li et al., 2013; Wang et al., 2015), in which both spatial and keyword conditions serve as Boolean filters. For example, a user may want to subscribe to tweets that are posted within 500 m of the user's office and that contain the terms "sales" and "clothing," to receive such tweets continuously.

Other kinds of subscription queries also exist. Specifically, top-k subscription queries return a subscriber only the top-k results, which are ranked by considering text relevance, spatial proximity, and the freshness of the object with respect to the query. For example, a subscription query may be submitted for a POI (e.g., a hotel) over streaming geo-tagged tweets, where the location of the POI is the query location and selected keywords from the POI's text description (e.g., service, cleanliness, free Wi-Fi) can be used as the query keywords. Such a subscription query may be of interest to a management team of a POI that wants to be continuously fed with the top-k tweets, ranked based on the their relevance to the query keywords, their proximity of tweets to the query location, and their freshness.

Each user may submit multiple subscription queries, and the number of Internet users is huge. Hence, the number of subscription queries may

be large. Thus focus has been on developing efficient solutions to handle large numbers of spatial keyword subscription queries over spatio-textual streams (Chen et al., 2013; Li et al., 2013; Mahmood et al., 2015; Wang et al., 2015; Chen et al., 2015).

The high-level idea of existing solutions is to group queries such that the queries in one group can be processed together over the spatio-textual data stream, rather than being processed individually, which is much more computationally expensive. Intuitively, similar queries should be grouped together so that good filtering conditions for a group can be designed to check whether a new spatio-textual object can be a result for some queries in the group. In addition to being able to efficiently process a large number of subscription queries over spatio-textual data streams where spatio-textual objects arrive at a high rate, such solutions must be capable of efficiently handling the arrival of new subscriptions and the discontinuation of existing subscriptions. The existing solutions also involve index structures to organize subscription queries for efficient query processing.

Exploring spatio-textual data There exist at least two types of research for exploring spatio-textual data. First, the region search problem aims to identify a region of a user-specified shape and size that maximizes or minimizes some aggregate score of the objects inside it (e.g., the sum of relevance of objects to a query), for user exploration. For certain aggregation scores like SUM, algorithms (Imai and Asano, 1983; Nandy and Bhattacharya, 1995; Choi et al., 2012) with complexity $O(n \log n)$ have been proposed, where n is the number of objects in $\mathcal{D}$. For more general aggregation function, such as submodular monotone score function, a solution with the worst-case complexity $O(n^2)$ exists (Feng et al., 2016). To improve the efficiency, an approximate algorithm is proposed to select a set T of spatial points from the space for representing spatial objects in $\mathcal{D}$. The selected points together preserve some properties of $\mathcal{D}$ such that the result region found on the set of points T can be an approximation of the result on $\mathcal{D}$ with performance guarantees (Feng et al., 2016).

Second, rather than finding a region that satisfies a user's needs, the problem of region exploration aims to explore and discover properties of user-specified regions. Given a user-specified region, one study (Skovsgaard et al., 2014) considers the problem of retrieving the top-k frequent words over the geo-textual data stream for the region. A new indexing technique is proposed for counting frequent items in static-sized summaries to allow the summaries to grow and shrink dynamically to adapt to changes in the incoming data. At query time, relevant summaries are merged to provide answers of the top-k frequent words with correctness guarantees.

Another example study of region exploration (Zhao et al., 2016) aims to efficiently discover topics in the spatio-textual data in a user-specified query region. Instead of learning a topic model for each query region, which is time-consuming, a two-phase approach to the exploratory topic mining task is advanced: (1) Indexing and Pre-Computation To support efficient online learning, the geographical space is partitioned into cells, and a grid-based index is constructed to organize the spatio-textual data. Then, by considering the memory constraint and accuracy guarantee of the online topic learning algorithm in the second phase, some cells are selected for learning latent Dirichlet allocation (LDA) models. (2) Online Topic Learning For a given query region, the pretrained topic models on the cells that overlap with the query region are combined as approximate topics for the spatio-textual data in the query region.

Examples of Application

Querying, mining, and exploring spatio-textual data have many applications. The basic types of spatial keyword queries are being supported in many online services that host large amounts of spatio-textual data, such as geo-tagged web pages, points of interest, geo-tagged tweets, and check-in. Other types of spatial keyword queries are designed to serve users for scenarios that

the basic queries are not sufficient. For example, consider a query with keywords "hotel, sports shop, and beach." Perhaps no nearby single place is a good match for all the keywords. Instead, a group of places that are close to each other and are close to the query location may together meet the user's needs better than any single object. Furthermore, it is expected that more applications will arise as more and more spatio-textual data is being generated in huge amount.

Querying streaming spatio-textual data is also very useful. For example, one application can be annotation of POIs with up-to-date geospatial social updates, such as geo-tagged tweets, as a manager of a POI may be interested in up-to-date tweets whose locations are close to the POI and whose text is relevant to the description of the POI. As another example, Groupon customers register their locations and keywords describing their interests. Then Groupon pushes Groupon messages to those customers that are near the messages and that have keywords that are relevant to the text of the messages.

Exploration of spatio-textual data is a desirable function as the amount of spatio-textual data continues to grow. The region research functionality can be used to find most influential regions for exploration, e.g., finding the best location for a billboard to influence as many consumers as possible to adopt a product, where the consumers may belong to a social network. Another example application of the region search is to find a region with most dense or diverse collection of services and attractions. One application of region exploration is to see what is happening in a region of interest.

Future Directions for Research

First, apart from spatial proximity and text relevance, many factors such as the quality of a spatio-textual object, click-throughs, and diversity can be considered in the ranking of spatial keyword query results. It is natural to consider whether these factors are useful for the ranking of query results and how important they are in ranking results. Reliable evaluation of ranking functions for spatio-textual data is essential to answer these questions. However, the utility of a result is dependent on the individual user. It is thus challenging to establish a reliable ground truth for the results of ranking queries.

Second, personalized location recommendation (Liu et al., 2017) aims to understand users' topical interests and mobility preferences from the users' historical data. Existing work on spatial keyword queries does not consider personalization, while personalized location recommendation does not consider spatial keyword search. It is of interest to attempt to bridge this gap, thus enabling personalized search on spatio-textual data.

Third, it is an open problem to effectively and efficiently support continuous queries on spatio-textual streams. For example, instead of receiving individual tweets from a stream, users may want to be notified in real time of relevant trending events or even of causal relationships among events. Furthermore, high-velocity spatio-textual data streams call for distributed systems to support querying and data analytics.

References

Cao X, Cong G, Jensen CS, Ooi BC (2011) Collective spatial keyword querying. In: SIGMOD, pp 373–384

Cao X, Cong G, Guo T, Jensen CS, Ooi BC (2015) Efficient processing of spatial group keyword queries. ACM Trans Database Syst 40(2):13

Chen Y, Suel T, Markowetz A (2006) Efficient query processing in geographic web search engines. In: SIGMOD, pp 277–288

Chen L, Cong G, Cao X (2013) An efficient query indexing mechanism for filtering geo-textual data. In: SIGMOD, pp 749–760

Chen L, Cong G, Cao X, Tan K (2015) Temporal spatial-keyword top-k publish/subscribe. In: ICDE, pp 255–266

Choi D-W, Chung C-W, Tao Y (2012) A scalable algorithm for maximizing range sum in spatial databases. Proc VLDB Endow 5(11):1088–1099

Christoforaki M, He J, Dimopoulos C, Markowetz A, Suel T (2011) Text vs. space: efficient geo-search query processing. In: CIKM, pp 423–432

Cong G, Jensen SC, Wu D (2009) Efficient retrieval of the top-k most relevant spatial web objects. In: PVLDB, pp 337–348

Cong G, Lu H, Ooi BC, Zhang D, Zhang M (2012) Efficient spatial keyword search in trajectory databases. CoRR, abs/(1205)2880

Felipe ID, Hristidis V, Rishe N (2008) Keyword search on spatial databases. In: ICDE, pp 656–665
Feng K, Cong G, Bhowmick SS, Peng W-C, Miao C (2016) Towards best region search for data exploration. In: SIGMOD. ACM
Guo T, Cao X, Cong G (2015) Efficient algorithms for answering the m-closest keywords query. In: SIGMOD, pp 405–418
Hariharan R, Hore B, Li C, Mehrotra S (2007) Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In: SSDBM, p 16
Hu H, Liu Y, Li G, Feng J, Tan K (2015) A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In: ICDE, pp 711–722
Imai H, Asano T (1983) Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. J Algorithms 4(4):310–323
Khodaei A, Shahabi C, Li C (2010) Hybrid indexing and seamless ranking of spatial and textual features of web documents. In: DEXA (1), pp 450–466
Li G, Wang Y, Wang T, Feng J (2013) Location-aware publish/subscribe. In: KDD, pp 802–810
Liu Y, Pham T, Cong G, Yuan Q (2017) An experimental evaluation of point-of-interest recommendation in location-based social networks. PVLDB 10(10): 1010–1021
Long C, Wong RC, Wang K, Fu WA (2013) Collective spatial keyword queries: a distance owner-driven approach. In: SIGMOD, pp 689–700
Mahmood AR, Aly AM, Qadah T, Rezig EK, Daghistani A, Madkour A, Abdelhamid AS, Hassan MS, Aref WG, Basalamah SM (2015) Tornado: a distributed spatio-textual stream processing system. PVLDB 8(12):2020–2023
Nandy SC, Bhattacharya BB (1995) A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. Comput Math Appl 29(8): 45–61
Rocha-Junior JB, Gkorgkas O, Jonassen S, Nørvåg K (2011) Efficient processing of top-k spatial keyword queries. In: SSTD, pp 205–222
Skovsgaard A, Sidlauskas D, Jensen CS (2014) Scalable top-k spatio-temporal term querying. In: ICDE, pp 148–159
Vaid S, Jones CB, Joho H, Sanderson M (2005) Spatio-textual indexing for geographical search on the web. In: SSTD, pp 218–235
Wang X, Zhang Y, Zhang W, Lin X, Wang W (2015) Ap-tree: efficiently support continuous spatial-keyword queries over stream. In: ICDE, pp 1107–1118
Wu D, Cong G, Jensen CS (2012a) A framework for efficient spatial web object retrieval. VLDB J 21(6): 797–822
Wu D, Yiu ML, Cong G, Jensen CS (2012b) Joint top-k spatial keyword query processing. IEEE Trans Knowl Data Eng 24(10):1889–1903
Zhang D, Chee YM, Mondal A, Tung AKH, Kitsuregawa M (2009) Keyword search in spatial databases: towards searching by document. In: ICDE, pp 688–699
Zhang D, Ooi BC, Tung AKH (2010) Locating mapped resources in web 2.0. In: ICDE, pp 521–532
Zhang C, Zhang Y, Zhang W, Lin X (2013a) Inverted linear quadtree: efficient top k spatial keyword search. In: ICDE, pp 901–912
Zhang D, Tan K-L, Tung AKH (2013b) Scalable top-k spatial keyword search. In: EDBT, pp 359–370
Zhou Y, Xie X, Wang C, Gong Y, Ma W-Y (2005) Hybrid index structures for location-based web search. In: CIKM, pp 155–162
Zhao K, Chen L, Cong G (2016) Topic exploration in spatio-temporal document collections. In: SIGMOD. ACM

Storage Failure

▸ Data Longevity and Compatibility

Storage Hierarchies for Big Data

Dirk Duellmann
Information Technology Department, CERN, Geneva, Switzerland

Motivation

Big data applications usually have to rely on a combination of storage media to achieve an economic balance between the capabilities of different media types and application needs.

Applications requirements vary significantly in data lifetime, number of concurrent data producers/consumers, fraction of active to passive data volume, sharing between parallel processing units, and the relative balance between CPU and IO requirements.

Storage media properties vary by several orders in price per capacity, latency for sequential and random access patterns, aggregate and single stream bandwidth, power requirements, endurance, and reliability. Several methods exist to further adapt these capabilities by combining several storage devices of the same type, but

larger and economically efficient setups are constructed by combining several different storage technologies.

In addition, larger storage deployments typically provide services to more than one application and hence aim to achieve an economic compromise between total cost of ownership and functional capabilities matching the ensemble of their applications.

Tiered Storage and HSM Systems

The above trade-off has since the mainframe days led to storage systems covering several layers in the traditional memory hierarchy (see Fig. 1) and integrate multiple technologies (e.g., disk and tape, disks of different capabilities) into tiered storage systems.

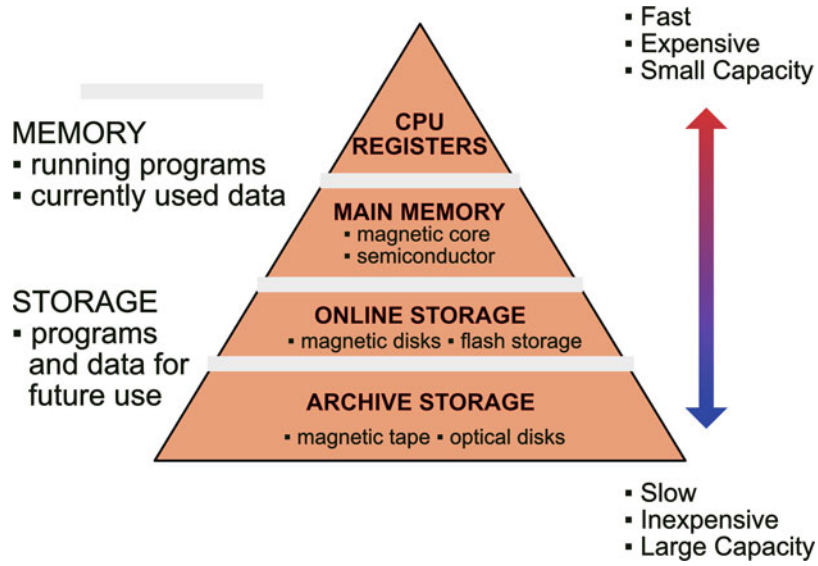
In addition to combining technology capabilities, these systems usually provide an abstraction for accessing data independent of the storage media and mechanisms allowing to move data between the different media.

Hierarchical storage management (HSM) systems extend the above abstraction and automatize the data movement between different media or quality of service based on the measured or predicted access patterns.

The resulting storage systems facilitate also the use of different storage technologies over the life cycle of larger data sets. Since they reduce the operator effort for data movements, they can diminish the risk of data loss due to human manipulation errors. Typical examples of HSM systems for data life cycle management are archive and backup systems, which manage access to one or more consistent data snapshots across different storage technologies.

Additional end-user benefits are a single consistent system interface and a stable abstraction from a, potentially evolving, back-end implementation of the storage system. This abstraction is particularly important in large shared storage installations, which usually need to operate without downtime and hence require the replacement of faulty components and storage media to take place with minimal impact on normal operation.

Automated data migration between storage technologies can conceptually take place at different data granularity – from small groups of data blocks within a user file to large groups of files that span multiple tape volumes. The dominating implementation choice though are systems that consider complete, individual data files as management granularity.



Storage Hierarchies for Big Data, Fig. 1 Memory and storage hierarchy

Hybrid Disk Drives and Multitiered File Systems

Conceptually similar, but at small volume scale, also hybrid disk drives implement a tiered storage system. These devices combine the fast random access characteristic of NAND-flash with the more economical storage capacity of magnetic disks in a single unit. Based on the measured access patterns, the device dynamically replicates often used data blocks in its small flash cache area, which can greatly improve application performance without adverse effects on volume cost or total unit capacity. These hybrid devices are most useful in deployments with space or connectivity constraints (e.g., laptops and small or embedded servers).

In contexts free of the above constraints, a similar hybrid combination can be constructed via multitier file systems such as FusionDrive (Apple Core Storage) or Hybrid Storage Pools in ZFS (Bonwick and Moore, 2003; Gregg, 2009) that combine the fast access of NAND-flash with the large capacity of magnetic disks.

Challenges for Hierarchical Storage Systems

HSM systems can greatly improve resource utilization in particular in environments with stable, organized workflows and established prioritization between concurrent applications.

In more complex environments, with many concurrent interdependent applications and many users with differing or conflicting priorities, HSM system can show visible limitations. In these cases both storage system operation and user experience become complex and less predictable – up to the point that the HSM automation benefits are out-weighted by additional human effort to stabilize operation (by manually partitioning the resources and throttling concurrent user activity) and artificial storage operations by the user side to "trick" the system into the desired behavior.

A further challenge in large-scale deployments can arise from a mismatch between the granularity of HSM data movements (e.g., individual files) and the conceptual entities of a user workflow (e.g., larger groups of files in a typical big data application). This granularity mismatch leads to difficulties let HSM data migrations appear as complete, atomic operations in the problem space of the user. A HSM with a purely file-based interface can only incomplete information of the user intention to access (or abandon) a larger group of files. The system may hence fail to properly predict the resource and time requirements to complete the intended operation or to predict the interference with data movements on behalf of other users/applications in a shared storage system.

Storage Hierarchies in Practical Applications

Tiered storage and hierarchical storage systems are today used in a variety of environments and are one of the key mechanisms to construct large-scale, cost-efficient storage systems that can adapt to changing access patterns over the lifetime of larger data set deployments. Data-intensive sciences such as high-energy physics, astrophysics, astronomy, and life sciences have constructed massive scientific data repositories (Bird et al., 2005; Pace, 2014) of several hundreds of petabytes by combining robotic tape archives, distributed disk clusters, and fast solid-state storage. The availability of reliable, high-bandwidth networks enabled to build large, distributed data repositories that enable collaborative scientific discovery and result sharing across dispersed communities. In data-intensive sciences, resource efficiency is of particular importance since computing budget constraints for storage media could limit the achievable statistical precision and hence relevance of a study.

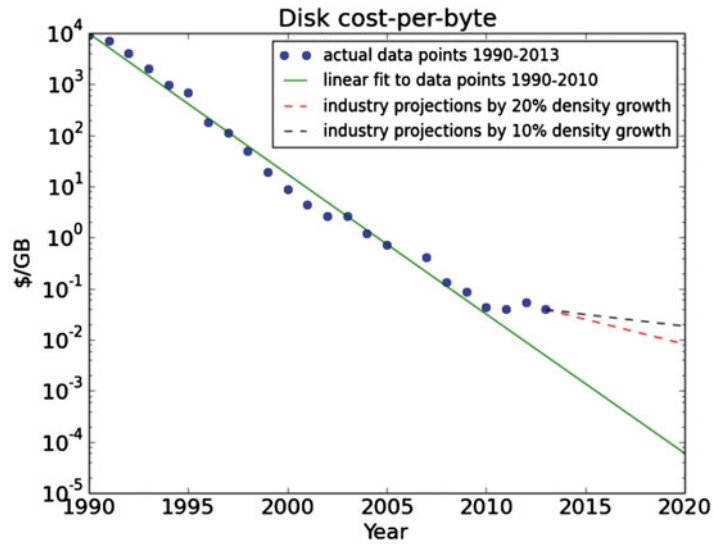
On the commercial side during the last decade, large data amounts from legacy sources and databases and an increasing volume of environmental or commercial sensors have become the basis for many analytical applications. Hadoop with HDFS as location-aware storage environment for parallel processing and S3 as archive storage with disk or tape as back-end have become increasingly popular. Recently also, the large available amount of directly

addressable memory has allowed to move previously big data analytical problems entirely into memory or nonvolatile solid-state storage to avoid the latency of magnetic media access and TCP networking round trips for much of their execution. In this context Apache Spark (Zaharia et al., 2010) plays an important role in combining parallel data selection on a large disk-based repository with large distributed in-memory caches in the same run-time environment.

Expected Technology and Market Changes

The established role of storage hierarchies in big data applications is expected to continue also in the future, when several new technologies will become available and market changes will shift the price balance between the available media types. On the magnetic disk side, one can since several years observe a slowdown of the areal media density evolution (see Fig. 2 from Kryder rate Walter, 2005; Gupta et al., 2014; Mellor, 2014; Klein, 2017) as the R&D effort increases to productize new magnetic recording techniques like shingled magnetic recording (Feldman and Gibson, 2013) (SMR), microwave-assisted magnetic recording (Zhu et al., 2007) (MAMR), and heat-assisted magnetic recording (Kryder et al., 2008) (HAMR).

Recording technologies such as SMR further imply a shifted balance between update and read performance, which will limit scalability in some use cases, unless combined with fast memory- or flash-based cache. Another influence on the media price balance may arise from the increasing consolidation of magnetic disk deployments to only a few, larger cloud providers, who are seeking increased influence on the design of magnetic disk units (Brewer et al., 2016). For the small number of vendors in the disk market, it will likely become more important to suit large computer center deployments than to maintain compatibility with the traditional form factor of previous desktop computer generations. The home and mobile markets are already largely based on devices with NAND-flash storage, which will see increasing competition from direct accessible persistent memory with higher durability and reduced write amplification (Intel 2015; https://arstechnica.com/information-technology/2017/03/intels-first-optane-ssd-375gb-that-you-can-also-use-as-ram/). This storage technology, if their advertised price point and functional capabilities are confirmed, would allow to further extend the problem space reachable by in-memory processing. Together with DRAM these new media would also enable



Storage Hierarchies for Big Data, Fig. 2 Magnetic disk price evolution

faster hybrid storage devices with NAND-flash or magnetic disks. Tape at the high volume end of the storage hierarchy has but due to very low media and power cost been remarkably successful – despite its for many applications problematic combination of high sequential rate and high access latency. Also here, visible market consolidation is influencing the continued technology evolution, and archive storage hierarchies involving tape will get increasing competition from disk- and flash-based alternatives (Gupta et al., 2014).

Conclusion

In conclusion, the use of tiered or hierarchical media combinations from several layers of the storage hierarchy will continue, and the recent fast, high-density, direct-access solid-state storage will allow to enlarge the problem size accessible to big in-memory applications. For problems that require larger volume scalability, the parallel cloud processing environments will benefit from some continued media density (and hence volume) growth but at lower Kryder rate. The performance gap between sequential read and other access methods (e.g., random access, write/update access) will continue to increase on the magnetic disk side. The benefits of integrating magnetic media with direct-access solid-state storage will therefore for many big data applications get even more important.

References

Bird I et al (2005) LHC computing grid—technical design report. CERN-LHCC-2005-024

Bonwick J, Moore B (2003) ZFS: the last word in file systems. http://opensolaris.org/os/community/zfs/docs/zfs_last.pdf

Brewer E et al (2016) Disks for data centers. https://research.google.com/pubs/pub44830.html

Feldman T, Gibson G (2013) Shingled magnetic recording—areal density increase requires new data management. Login 38(3):22–30

Gregg B (2009) Hybrid storage pool: top speeds. http://dtrace.org/blogs/brendan/2009/10/08/hybrid-storage-pool-top-speeds/

Gupta P et al (2014) An economic perspective of disk vs. flash media in archival storage. In: IEEE MASCOTS 2014

Intel (2015) Micron debut 3D XPoint storage technology 1,000x faster than current SSDs. https://www.cnet.com/news/intel-and-micron-debut-3d-xpoint-storage-technology-thats-1000-times-faster-than-existing-drives/

Klein A (2017) Hard disk cost per gigabyte. https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/

Kryder et al (2008) Heat assisted magnetic recording. Proc IEEE 96(11):1810–1835

Mellor C (2014) Kryder's law craps out: race to UBER-CHEAP STORAGE is OVER. https://www.theregister.co.uk/2014/11/10/kryders_law_of_ever_cheaper_storage_disproven/

Pace A (2014) Technologies for large data management in scientific computing. Int J Mod Phy C 25(2):1430001

Walter C (2005) Kryder's law. Sci Am 293(2):32–3

Zaharia M et al (2010) Spark: cluster computing with working sets. Technical report No. UCB/EECS-2010-53, University of California, Berkeley

Zhu J, Zhu X, Tang Y (2007) Microwave Assisted Magnetic Recording, IDEMA

Storage Technologies for Big Data

Zhaojie Niu and Bingsheng He
National University of Singapore, Singapore, Singapore

Overview

This article introduces big data storage systems. It first describe the development of storage techniques for big data. Then, it compares the different storage models and their most suitable use cases. Finally, it talks about the impact of emerging hardware on the big data storage techniques and introduces some hot research topics.

Introduction and Background

In the big data era, data are increasingly gathered from heterogeneous devices like IoT devices, sensor networks, scientific experiments, websites, and many other applications producing data in various formats. The data sets are

so voluminous and complex that traditional relational databases are inadequate to deal with them. Thus, innovative research and development of storage systems that can provide highly scalable, reliable, and efficient storage for dynamically increasing data is required. A movement to NoSQL (Not only SQL) occurs, and many NoSQL databases are proposed. They have simplicity of design, simpler "horizontal" scaling to clusters consists of thousands of servers, and more fine-grained control over availability. The data structures used by NoSQL databases are optimized for target data formats and sources, which are different from those used in relational databases, making some operations faster in NoSQL. Many NoSQL stores compromise consistency (in the context of CAP theory (Seth and Nancy, 2002)) in favor of partition tolerance, availability, and performance. These NoSQL stores also lack true ACID transaction supported by traditional relational databases, although a few databases, such as Google spanner (David et al., 2017), start to embrace the relational model.

Relational databases have been widely used efficiently for transaction processing in terms of storage and processing for many decades. The relational databases support ACID transaction that also limits their horizontal scalability. To achieve availability and better scalability, most NoSQL databases compromise consistency and do not support true ACID transaction support as traditional relational databases. The strengths and weakness for traditional relational database and NoSQL database are summarized in Table 1.

The NoSQL databases mainly consists four types of data models: key-value, column-oriented, document-oriented, and graph. These NOSQL systems are being used to handle exponentially growing data but cannot tolerate the performance degradation caused by data persistence. On the other hand, these systems face the challenges posed by the inherent difficulties in scaling DRAM and the cost of the DRAM. To address these challenges, new emerging hardware technologies like nonvolatile main memory (NVRAM) have lately received a great deal of attention in the database community. NVRAM is positioned between DRAM and secondary storage (such as SSD), both in terms of performance and cost. NVRAM can fundamentally change in-memory databases as data structures do not have to be explicitly backed up to hard drives or SSDs but can be inherently persistent in main memory (David et al., 2015; Jasmina et al., 2015; Mihnea et al., 2017; Sudarsun et al., 2016; Colaso et al., 2017). In section "Taxonomy of Big Data Storage Technologies", we introduce these NoSQL systems based on the data model and how the emerging hardware impacts the design of the NOSQL systems.

Taxonomy of Big Data Storage Technologies

NoSQL databases mainly consist four types of data models: key-value, column-oriented,

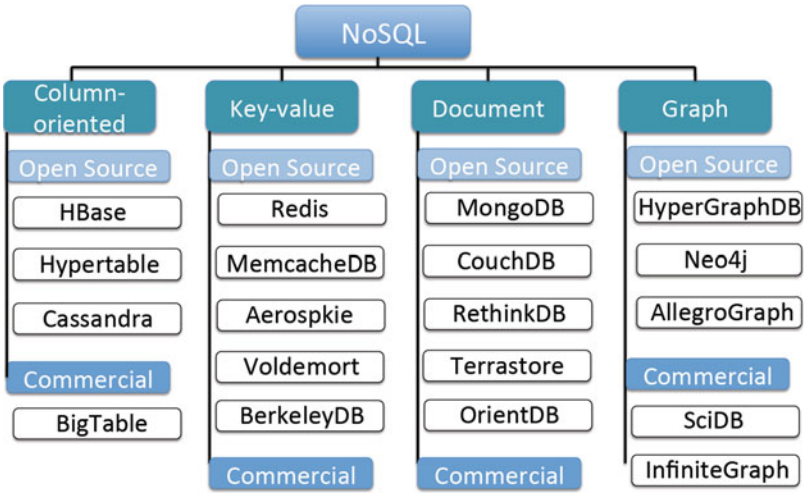
Storage Technologies for Big Data, Table 1 Strengths and weakness for traditional relational database and NoSQL database

DB	Strength	Weakness
Traditional relational database	Support highly structured data; Efficient storage and processing in auxiliary server; ACID transaction support; Vertical scalability; Specialized data manipulation languages; Specialized data schema	Performance and processing delay bottleneck with growth of data; ACID support which hinders scalability; Limitation for schema-less and unstructured data
NoSQL database	Support heterogeneous structured data; Horizontal scalability with extendible commodity servers; High reliability; High availability	No compliance with ACID in order to achieve high scalability and high availability for most NoSQL databases

Storage Technologies for Big Data, Table 2
Performance, scalability, flexibility and complexity of different data models for NoSQL databases

Data model	Performance	Scalability	Flexibility	Complexity
Key-value	High	High	High	None
Column-oriented	High	High	Moderate	Low
Document-oriented	High	Variable	High	Low
Graph	Variable	Variable	High	High
Relational	Variable	Variable	Low	Moderate

Storage Technologies for Big Data, Fig. 1
Taxonomy of big data storage technologies based on data models



document-oriented, and graph (Michael, 2014). The performance, scalability, flexibility, and complexity for different data models are different that is shown in Table 2. We classify NoSQL databases based on the data models and introduce some representing systems for each category.

We classify the current representing big data storage system according to data models as shown in Fig. 1 and introduce them one by one in detail. We also discuss the recent development on emerging storage techniques with nonvolatile memory (NVRAM).

Key-Value Data Model

The key-value type, basically, uses a hash table in which there exist a unique key and a pointer to a particular item of data. A bucket is a logical group of keys, but they don't physically group the data. There can be identical keys in different buckets. This in-memory data structure design needs to explicitly backup in case of system crashes in DRAM which degrades the performance. NVRAM guarantees data persistency even across power outages while it comes with new challenges as the consistency and persistency of in-memory data structures have to be guaranteed even across system crashes. Data structure and algorithms of the hash table especially on NVRAM need to be redesigned for the use in NV-persisted in-memory databases (David et al., 2015).

Column-Oriented Data Model

In column-oriented NoSQL database, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain a virtually unlimited number of columns that can be created at runtime or the definition of the schema. Read and write is done using columns rather than rows. To increase the scalability and reduce the cost of using DRAM, SAP starts to explore the new column-based in-memory data structures and efficient algorithms

which are directly stored and used in NVRAM blocks (Mihnea et al., 2017).

Document-Oriented Data Model

The data which is a collection of key-value pairs is compressed as a document store quite similar to a key-value store, but the only difference is that the values stored (referred to as "documents") provide some structure and encoding of the managed data. XML, JSON (Java Script Object Notation), and BSON (which is a binary encoding of JSON objects) are some common standard encodings. In order to provide efficient memory access for these objects, cost-effective memory placement mechanisms are proposed on NVRAM (Sudarsun et al., 2016).

Graph Data Model

In a Graph Base NoSQL database, you will not find the rigid format of SQL or the tables and columns representation; a flexible graphical representation is instead used which is perfect to address scalability concerns. Graph structures are used with edges, nodes, and properties which provide index-free adjacency. Data can be easily transformed from one model to the other using a Graph Base NoSQL database. With the increase of the scale of the graph, the capacity of scaling of DRAM is not able to satisfy the requirements of analytics over graph with trillions of connections. Explores of replacing DRAM with emerging nonvolatile memories (NVRAM) on large-scale graph analytics framework are performed. Some hybrid memory systems with NVRAM and DRAM are proposed in a cost-effective manner and the performance it can achieve is close to DRAM-only performance (Jasmina et al., 2015).

References

Andrei M, Lemke C, Radestock G, Schulze R, Thiel C, Blanco R, Meghlan A, Sharique M, Seifert S, Vishnoi S, Booss D, Peh T, Schreter I, Thesing W, Wagle M, Willhalm T (2017) Sap HANA adoption of non-volatile memory. Proc VLDB Endow 10(12):1754–1765

Bacon DF, Bales N, Bruno N, Cooper BF, Dickinson A, Fikes A, Fraser C, Gubarev A, Joshi M, Kogan E, Lloyd A, Melnik S, Rao R, Shue D, Taylor C, van der Holst M, Woodford D (2017) Spanner: becoming a SQL system. In: Proceedings of the 2017 ACM international conference on management of data (SIGMOD'17). ACM, pp 331–343

Colaso A, Prieto P, Abad P, Herrero JA, Menezo LG, Puente V, Gregorio JA (2017) Memory hierarchy characterization of NoSQL applications through full-system simulation. IEEE Trans Parallel Distrib Syst PP(99):1–1

Gilbert S, Lynch N (2002) Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2):51–59

Kannan S, Gavrilovska A, Schwan K (2016) PVM: persistent virtual memory for efficient capacity scaling and object storage. In: Proceedings of the eleventh European conference on computer systems (EuroSys'16). ACM, pp 13:1–13:16

Malicevic J, Dulloor S, Sundaram N, Satish N, Jackson J, Zwaenepoel W (2015) Exploiting NVM in large-scale graph analytics. In: Proceedings of the 3rd workshop on interactions of NVM/FLASH with operating systems and workloads (INFLOW'15). ACM, pp 2:1–2:9

Mior MJ (2014) Automated schema design for nosql databases. In: Proceedings of the 2014 SIGMOD PhD symposium (SIGMOD'14). ACM, pp 41–45

Schwalb D, Dreseler M, Uflacker M, Plattner H (2015) Nvc-hashmap: a persistent and concurrent hashmap for non-volatile memories. In: Proceedings of the 3rd VLDB workshop on in-memory data mangement and analytics (IMDM'15). ACM, pp 4:1–4:8

Straight-Line Context-Free Tree Grammars

▸ Grammar-Based Compression

Straight-Line Programs

▸ Grammar-Based Compression

Stratosphere Platform

▸ Apache Flink

Stream Benchmarks

Jeyhun Karimov
DFKI GmbH, Berlin, Germany

Synonyms

Stream performance evaluation

Definitions

Stream benchmarks deal with performance evaluation techniques and define related metrics for stream data processing systems.

Overview

Firstly, we discuss background information on database benchmarking, foundations, main metrics, and main features for stream data benchmarking. Then, we provide related stream benchmarks and categorize them with respect to the application area.

Historical Background

Lee et al. (1997) initiated one of the first works in the related area, MediaBench, by evaluating and synthesizing multimedia and communications systems. Abadi et al. (2003) and Motwani et al. (2003) pioneered one of the first stream data processing systems, Aurora and STREAM, respectively. The need to compare the performance characteristics of streaming systems relative to each other and to alternative (e.g., Relational Database) systems endorsed the development of Linear Road benchmark (Arasu et al., 2004).

Foundations

Stream data processing is key when the large volumes of input data have to be processed fast to quickly adapt and react to changes. Therefore, stream data processing has gained significant attention.

The main intuition behind stream benchmarks is to define a standard to compare streaming systems, which has different characteristics, in a various use-cases. Stream benchmarks simulate an environment with different workloads and analyze the behavior of the systems to be tested. The more similar the benchmark to the real production environment, the more realistic and valuable it is.

In the high level, stream benchmark framework consists of two main components: system under test and driver. Driver is responsible for simulating the production environment w.r.t. a given use-case. System under test is actual tested streaming system which can be out-of-the-box or tuned.

Accurately representing the system under test is important when designing benchmarks. Schroeder et al. (2006) categorize benchmarks into closed, open, and partly open models. In a closed system model, the input arrivals are triggered after the completion of the previous input processing and some thinking time delay. In an open system model, on the other hand, new input arrivals and the completion of the previous input processing are independent. In a partly open system model, we specify the settings for which partly open model behaves like a closed or an open model.

Below, we categorize existing stream benchmarks in literature. Firstly, we talk about the main metrics in stream benchmarks and their definitions. Secondly, we analyze stream benchmarks which concentrate on specific features such as fault tolerance and state management. Lastly, we analyze stream benchmarks built for specific industrial use-cases.

Metrics

Main metrics for stream benchmarks are latency and throughput. Achieving high throughput while preserving low latency is the main goal for streaming systems.

Latency

Defining a standard latency metric definition for streaming systems is a challenging task. One reason is significant architectural and computational differences among stream data processing engines. Another reason is calculating latency for stateful operators in nontrivial.

Chintapalli et al. (2016a) use Redis for stateful computations as part of the system under test. The authors calculate the event-time latency by subtracting the window start time and duration time from the last updated time of a particular input record. Perera et al. (2016) propose reproducible benchmarks for Apache Spark and Flink on public clouds. The authors do not use a standard latency definition, as, in this case, the latency measurement is experiment and application specific. Lu et al. (2014) propose a new stream benchmark by separating the data generator and system under test. The authors put a mediator layer between the two components. They define latency as an average time span from the arrival of a record till the end of processing of the record. Qian et al. (2016) also adopted a similar approach. Karimov et al. (2018) develop stream benchmark framework that overcome overhead of a mediator layer. The authors show how processing-time latency might mislead when compared with event-time latency.

Throughput

Throughput is another essential metric for stream data processing systems. Similar to latency, measuring and defining a standard throughput metric for all streaming systems is nontrivial.

Chintapalli et al. (2016a) separate the data generator and system under test with an intermediate layer between them. The authors calculate the throughput by configuring the speed of data generator for a specific workload. Lopez et al. (2016a), on the other hand, rely on Kafka's sampled throughput rates. Lu et al. (2014) measure the overall system under test throughput and throughput per node. The authors define overall throughput metric by count (average count of records per second) and size (average data size in terms of bytes processed per second). Shukla and Simmhan (2016) define throughput as the rate of output tuples emitted from the output operators in a unit time. Dayarathna and Suzumura (2013) define job throughput in two ways. First, the authors measure the time required to process a specific amount of events. Second, the authors measure the number of tuples processed in a given amount of time. The throughput computation is performed based on both time periods. Samosir et al. (2016), on the other hand, adopt the throughput metric used in batch processing systems. Karimov et al. (2018) propose maximum sustainable throughput throughout the whole experiment.

Benchmarking the energy consumption of stream data processing engines is another important aspect of stream benchmarks. Dayarathna et al. (2017) adopt Linear Road benchmark for testing the energy efficiency of S4, Storm, ActiveMQ, Esper, Kafka, and Spark Streaming. The key finding of this work is that better power consumption behaviors in the context of data stream processing systems can be achieved by setting tuple sizes to be moderate and scheduling plans to have balanced system overhead.

Features

Besides focusing on metrics computations, stream benchmarks also concentrate on specific features of stream data processing engines.

Fault Tolerance

Lopez et al. (2016a) study streaming systems' tolerance to failures by analyzing the system behavior after detecting the failure. The system behavior includes the message losses and latency/throughput change during node failure. Gradvohl et al. (2014) categorize fault tolerance behavior in stream data processing systems into replication components, upstream backup, checkpoint, and recovery and analyze the system utilization of these strategies. Mohamed et al. (2017) propose a driver which allows programmatic specification of complex fault scenarios. Chauhan et al. (2012) measure the systems' tolerance to faults by (i) measuring the

number of events handled and (ii) checking the number of events that were missed when nodes go down in cluster. Qian et al. (2016) adopt identity workload and consider only one node failure at a time. The benchmark suite collects the performance metrics in node-failure workload and compares it with non-faulty workload.

State Management

Linear Road benchmark, Arasu et al. (2004) and Jain et al. (2006), consists of continuous queries which update operator state by processing the incoming stream. Kipf et al. (2017) analyze the limitation of efficiently exposing the state to analytical queries for stream data processing systems. The authors compare main-memory databases and streaming engines and propose new methods to advance state management in streaming systems.

Key Applications

In this section we categorize stream benchmarks based on different industrial use-cases.

Data mining. Zhang et al. (2012) and Le-Phuoc et al. (2012) are the pioneers to propose SRBench and LSBench, the first benchmarks for RDF streaming and Linked Stream Data processing. The authors adopt wide range of queries including simple graph pattern matching queries and the ones with complex reasoning tasks. DellAglio et al. (2013) propose CSRBench, an extension for SRBench. The authors overcome the main shortcoming of SRBench and LSBench, which is inability to assess the correctness of query evaluation results, by analyzing the operational semantics of the particular processors. Ali et al. (2015) address another limitation of SRBench and LSBench, addressing the dynamic application requirements and data-dependent properties. The authors propose the workloads which include fluctuating streaming rates during query execution and changing the application requirements over a some time duration. Implementing, modeling, and evaluating the provisioning algorithms for stream processing applications is another related work, in which authors propose VISP Testbed (Hochreiner, 2017). The toolkit provides a common runtime for stream processing applications.

E-commerce. Teng et al. (2017) analyze streaming system behavior in e-commerce scenarios. The authors provide a data generator, as part of the benchmark suite, with certain user models, which adopt a certain user habits in e-commerce platforms. Tucker et al. (2008) propose NEXMark, an extension of XMark, Schmidt et al. (2001), based on online auction system. Currently, NEXMark is used as a benchmark suite in Apache Beam, Buzzwords (2017).

IoT. Shukla and Simmhan (2016) develop a benchmark suite for streaming systems for IoT applications. The authors classify 13 common IoT tasks with functional categories. Moreover, the benchmark suite provides two IoT applications being statistical summarization and predictive analytics. CityBench is the benchmark to evaluate RDF stream processing systems in IoT scenarios, Ali et al. (2015). The authors use traffic vehicles, parking, weather, pollution, cultural, and library events, with changing event rates and playback speeds as part of the data generator. Shukla et al. (2017) extend the existing stream benchmarks in IoT proposing RIoTBench. The benchmark includes 27 common IoT tasks. Moreover, the authors propose four IoT application benchmarks composed from the proposed tasks.

Network. Nazhandali et al. (2005) propose stream benchmark for sensor network systems, suitable for sensor processors. The authors propose new metrics being EPB (Energy Per Bundle) and CFP (Composition Footprint) to evaluate and compare systems under test. Lopez et al. (2016b) analyze the performance of Virtualized Network Function for real-time thread detection using stream processing. Wolf and Franklin (2000) propose telecommunication benchmark for network processors. The authors adopt four workloads for data stream processing in telecommunications scenario. Trivedi et al.

(2016) analyze yet another interesting aspect, the (ir)relevance of network bandwidth to modern streaming engines. The key finding of this paper is that current streaming engines need significant architectural improvements as they cannot benefit from high bandwidth networks.

Multi-core processors. Zhang et al. (2017) benchmark the current design of stream data processing engines on multi-core processors analyzing the possible bottlenecks of massively parallel JVM-based streaming engines.

CEP. Mendes et al. (2009) were among the pioneers to propose a benchmark for CEP systems. The authors provide series of queries to exercise factors such as window size and policy, selectivity, and event dimensionality. Mendes et al. (2013) propose BiCEP, the domain-specific benchmark suite, to evaluate different performance aspects of event processing platforms. Alevizos and Artikis (2014) adopt existing techniques to analyze widely used Esper system which employs a SQL-based language and RTEC which is a dialect of the Event Calculus.

Machine learning. Gama et al. (2009) propose a general framework for assessing predictive stream learning algorithms. Gama et al. (2013) focus on decision models and develop a benchmark suite to evaluate continuously evolving streaming and to detect and react to real-time input data. Imai et al. (2017) utilize a machine learning model to predict the maximum sustainable throughput in streaming systems.

Bottlenecks. When designing a system, it is important to detect and avoid bottlenecks. For streaming systems, Chintapalli et al. (2016b) show the Zookeeper being a main performance bottleneck for Storm. For stream benchmarks, Friedrich et al. (2017) show the limitations of existing benchmark designs and the possible biased results. Moreover, Artisans (2017) makes a disclaimer for the original implementation of Chintapalli et al. (2016a), showing an intermediate message layer, Kafka, and external state management system, Redis, are actually being a bottleneck for Flink's overall performance.

Cross-References

▸ Distributed Systems

References

Abadi DJ, Carney D, Çetintemel U, Cherniack M, Convey C, Lee S, Stonebraker M, Tatbul N, Zdonik S (2003) Aurora: a new model and architecture for data stream management. VLDB J Int J Very Large Data Bases 12(2):120–139

Alevizos E, Artikis A (2014) Being logical or going with the flow? A comparison of complex event processing systems. In: SETN. Springer, pp 460–474

Ali MI, Gao F, Mileo A (2015) Citybench: a configurable benchmark to evaluate RSP engines using smart city datasets. In: International semantic web conference. Springer, pp 374–389

Arasu A, Cherniack M, Galvez E, Maier D, Maskey AS, Ryvkina E, Stonebraker M, Tibbetts R (2004) Linear road: a stream data management benchmark. In: Proceedings of the thirtieth international conference on very large data bases, vol 30. VLDB Endowment, pp 480–491

Artisans D (2017) Extending the Yahoo! streaming benchmark. https://data-artisans.com/blog/extending-the-yahoo-streaming-benchmark. Online: Accessed 1 Nov 2017

Buzzwords B (2017) Nexmark: using apache beam to create a unified benchmarking suite. https://berlinbuzzwords.de/sites/berlinbuzzwords.de/files/media/documents/nexmark_suite_for_apache_beam_berlin_buzzwords_1.pdf. Online: Accessed 1 Nov 2017

Chauhan J, Chowdhury SA, Makaroff D (2012) Performance evaluation of yahoo! s4: a first look. In: 2012 seventh international conference on P2P, parallel, grid, cloud and internet computing (3PGCIC). IEEE, pp 58–65

Chintapalli S, Dagit D, Evans B, Farivar R, Graves T, Holderbaugh M, Liu Z, Nusbaum K, Patil K, Peng BJ et al (2016a) Benchmarking streaming computation engines: storm, flink and spark streaming. In: 2016 IEEE international parallel and distributed processing symposium workshops. IEEE, pp 1789–1792

Chintapalli S, Dagit D, Evans R, Farivar R, Liu Z, Nusbaum K, Patil K, Peng B (2016b) Pacemaker: when zookeeper arteries get clogged in storm clusters. In: 2016 IEEE 9th international conference on cloud computing (CLOUD). IEEE, pp 448–455

Dayarathna M, Suzumura T (2013) A performance analysis of system s, s4, and esper via two level benchmarking. In: International conference on quantitative evaluation of systems. Springer, pp 225–240

Dayarathna M, Li Y, Wen Y, Fan R (2017) Energy consumption analysis of data stream processing: a benchmarking approach. Softw Pract Exp 47(10): 1443–1462

DellAglio D, Calbimonte JP, Balduini M, Corcho O, Della Valle E (2013) On correctness in RDF stream processor benchmarking. In: International semantic web conference. Springer, pp 326–342

Friedrich S, Wingerath W, Ritter N (2017) Coordinated omission in NoSQL database benchmarking. In: BTW (Workshops), pp 215–225

Gama J, Sebastião R, Rodrigues PP (2009) Issues in evaluation of stream learning algorithms. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 329–338

Gama J, Sebastião R, Rodrigues PP (2013) On evaluating stream learning algorithms. Mach Learn 90(3): 317–346

Gradvohl ALS, Senger H, Arantes L, Sens P (2014) Comparing distributed online stream processing systems considering fault tolerance issues. J Emerg Technol Web Intell 6(2):174–179

Hochreiner C (2017) Visp testbed-a toolkit for modeling and evaluating resource provisioning algorithms for stream processing applications. Strategies 1(6):9–17

Imai S, Patterson S, Varela CA (2017) Maximum sustainable throughput prediction for data stream processing over public clouds. In: Proceedings of the 17th IEEE/ACM international symposium on cluster, cloud and grid computing. IEEE Press, pp 504–513

Jain N, Amini L, Andrade H, King R, Park Y, Selo P, Venkatramani C (2006) Design, implementation, and evaluation of the linear road benchmark on the stream processing core. In: Proceedings of the 2006 ACM SIGMOD international conference on Management of data. ACM, pp 431–442

Karimov J, Rabl T, Katsifodimos A, Samarev R, Heiskanen H, Markl V (2018) Benchmarking distributed stream data processing systems. In: ICDE

Kipf A, Pandey V, Böttcher J, Braun L, Neumann T, Kemper A (2017) Analytics on fast data: main-memory database systems versus modern streaming systems. In: EDBT, pp 49–60

Le-Phuoc D, Dao-Tran M, Pham MD, Boncz P, Eiter T, Fink M (2012) Linked stream data processing engines: facts and figures. In: The semantic Web–ISWC 2012, pp 300–312

Lee C, Potkonjak M, Mangione-Smith WH (1997) Mediabench: a tool for evaluating and synthesizing multimedia and communications systems. In: Proceedings of the 30th annual ACM/IEEE international symposium on microarchitecture. IEEE Computer Society, pp 330–335

Lopez MA, Lobato AGP, Duarte OCM (2016a) A performance comparison of open-source stream processing platforms. In: 2016 IEEE global communications conference (GLOBECOM). IEEE, pp 1–6

Lopez MA, Lobato AGP, Duarte OCM, Pujolle G (2016b) Design and performance evaluation of a virtualized network function for real-time threat detection using stream processing

Lu R, Wu G, Xie B, Hu J (2014) Stream bench: towards benchmarking modern distributed stream computing frameworks. In: 2014 IEEE/ACM 7th international conference on utility and cloud computing (UCC). IEEE, pp 69–78

Mendes MR, Bizarro P, Marques P (2009) A performance study of event processing systems. In: Technology conference on performance evaluation and benchmarking. Springer, pp 221–236

Mendes M, Bizarro P, Marques P (2013) Towards a standard event processing benchmark. In: Proceedings of the 4th ACM/SPEC international conference on performance engineering. ACM, pp 307–310

Mohamed S, Forshaw M, Thomas N, Dinn A (2017) Performance and dependability evaluation of distributed event-based systems: a dynamic code-injection approach. In: Proceedings of the 8th ACM/SPEC on international conference on performance engineering. ACM, pp 349–352

Motwani R, Widom J, Arasu A, Babcock B, Babu S, Datar M, Manku G, Olston C, Rosenstein J, Varma R (2003) Query processing, resource management, and approximation in a data stream management system. In: CIDR

Nazhandali L, Minuth M, Austin T (2005) Sensebench: toward an accurate evaluation of sensor network processors. In: Proceedings of the IEEE international workload characterization symposium, 2005. IEEE, pp 197–203

Perera S, Perera A, Hakimzadeh K (2016) Reproducible experiments for comparing apache flink and apache spark on public clouds. arXiv preprint arXiv:161004493

Qian S, Wu G, Huang J, Das T (2016) Benchmarking modern distributed streaming platforms. In: 2016 IEEE international conference on industrial technology (ICIT). IEEE, pp 592–598

Samosir J, Indrawan-Santiago M, Haghighi PD (2016) An evaluation of data stream processing systems for data driven applications. Proc Comput Sci 80:439–449

Schmidt AR, Waas F, Kersten ML, Florescu D, Manolescu I, Carey MJ, Busse R (2001) The XML benchmark project

Schroeder B, Wierman A, Harchol-Balter M (2006) Open versus closed: a cautionary tale. In: NSDI, vol 6, pp 18–18

Shukla A, Simmhan Y (2016) Benchmarking distributed stream processing platforms for IoT applications. In: Technology conference on performance evaluation and benchmarking. Springer, pp 90–106

Shukla A, Chaturvedi S, Simmhan Y (2017) Riotbench: a real-time IoT benchmark for distributed stream processing platforms. arXiv preprint arXiv:170108530

Teng M, Sun Q, Deng B, Sun L, Qin X (2017) A tool of benchmarking realtime analysis for massive behavior data. In: Asia-Pacific web (APWeb) and web-age information management (WAIM) joint conference on web and big data. Springer, pp 345–348

Trivedi A, Stuedi P, Pfefferle J, Stoica R, Metzler B, Koltsidas I, Ioannou N (2016) On the [ir] relevance of network performance for data processing. Network 40:60

Tucker P, Tufte K, Papadimos V, Maier D (2008) Nexmark–a benchmark for queries over data streams (draft). Technical report, OGI School of Science & Engineering at OHSU

Wolf T, Franklin M (2000) Commbench-a telecommunications benchmark for network processors. In: 2000 IEEE international symposium on performance analysis of systems and software (ISPASS). IEEE, pp 154–162

Zhang Y, Duc PM, Corcho O, Calbimonte JP (2012) Srbench: a streaming RDF/SPARQL benchmark. In: International semantic web conference. Springer, pp 641–657

Zhang S, He B, Dahlmeier D, Zhou AC, Heinze T (2017) Revisiting the design of data stream processing systems on multi-core processors. In: 2017 IEEE 33rd international conference on data engineering (ICDE). IEEE, pp 659–670

Stream Learning

- ▸ Online Machine Learning Algorithms over Data Streams
- ▸ Online Machine Learning in Big Data Streams: Overview
- ▸ Recommender Systems over Data Streams
- ▸ Reinforcement Learning, Unsupervised Methods, and Concept Drift in Stream Learning

Stream Performance Evaluation

- ▸ Stream Benchmarks

Stream Processing Language

- ▸ Stream Processing Languages and Abstractions

Stream Processing Languages and Abstractions

Martin Hirzel and Guillaume Baudart
IBM Research AI, Yorktown Heights, NY, USA

Synonyms

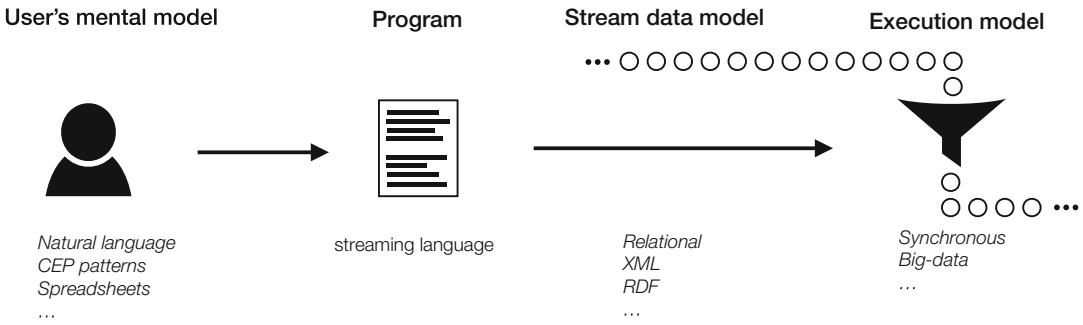
Continuous dataflow language; Streaming language; Stream processing language

Definitions

A *stream processing language* is a programming language for specifying streaming applications. Here, a *stream* is an unbounded sequence of data items, and a *streaming application* is a computer program that continuously consumes input streams and produces output streams. This article surveys recent streaming languages designed around the user's mental model, the stream data model, or the execution model, as illustrated in Fig. 1. In addition to specific languages, this article also discusses *abstractions* for stream processing, which are high-level language constructs that make it easy to express common stream processing tasks.

Overview

Continuous data streams arise from many directions, including sensors, communications, and commerce. Stream processing helps when low-latency responses are of the essence or when streams are too big to store for offline analysis. Programmers can of course write streaming applications in a general-purpose language without resorting to a dedicated domain-specific language (DSL) for streaming. However, using a streaming language makes code easier to read, write, understand, reason about, modularize, and optimize. Indeed, a suitable streaming language can help developers conceive of a solution to their streaming problems.



Stream Processing Languages and Abstractions, Fig. 1 Stream processing languages

This article provides definitions, surveys concepts, and offers pointers for more in-depth study of recent streaming languages. The interested reader may also want to refer to earlier papers for historic perspective: the 1997 survey by Stephens focuses on streaming languages (Stephens, 1997), the 2002 survey by Babcock et al. focuses on approximate streaming algorithms (Babcock et al., 2002), and the 2004 survey by Johnston et al. addresses dataflow languages, where streaming is a special case of dataflow (Johnston et al., 2004).

The central abstractions of stream processing are streams, operators, and stream graphs. A *stream* is an unbounded sequence of data items, for example, position readings from a delivery truck. A streaming *operator* is a stream transformer that transforms input streams to output streams. From the perspective of a streaming application, an operator can also have zero input streams (*source*) or zero output streams (*sink*). Finally, a *stream graph* is a directed graph whose nodes are operators and whose edges are streams. Some literature assumes that the shape of stream graphs is restricted, e.g., acyclic, but this article makes no such assumption. While only some streaming languages make the stream graph explicit, others use it as an intermediate representation. For example, the query plan generated from streaming SQL dialects is a stream graph.

The field of streaming languages is diverse and fast-moving. To understand where that diversity comes from, it is instructional to classify streaming languages by their raison d'être. Some streaming languages are based on the attitude that since streams are data in motion, data is most central, and the language should be built around a data model (relational, XML, RDF). Other streaming languages focus on the execution model for processing the dataflows efficiently, by enforcing timing constraints or exploiting distributed hardware (synchronous, big-data). A third class of streaming languages focus more on enabling the end user to develop streaming applications in high-level or familiar abstractions (complex events, spreadsheets, or even natural language). Section "Findings" surveys languages in each of these classes, and section "Examples" gives concrete examples for two languages.

Findings

This section gives an overview of the field of stream processing languages by surveying eight prominent approaches. Each approach is exemplified by one concrete language. The approaches are grouped along the lines of the previous section into approaches driven by the data model, by the execution model, or by the target user.

Data-Model Driven Streaming Languages

The success of the **relational** data model for database systems has inspired streaming dialects of the SQL database language. These dialects benefit from developers' familiarity with SQL and from its relational algebra underpinnings. A prominent example is the CQL language, which complements standard relational operators with operators to transform streams into relations and

vice versa (Arasu et al., 2006). CQL lends itself to strong static typing, cf. Figure 10 of Soulé et al. (2016). Efforts toward standardizing streaming SQL focused on clarifying semantic corner cases (Jain et al., 2008).

The success of **XML** as a universal exchange format for events and messages has inspired XML-based streaming languages. These languages take advantage of a rich ecosystem of XML tools and standards and of the fact that XML documents are self-describing. The languages come in different flavors, from view maintenance over XML updates in NiagaraCQ (Chen et al., 2000) to languages that process streams where each data item is a (part of an) XML document (Diao et al., 2002; Mendell et al., 2012).

The Resource Description Framework (**RDF**) is a versatile data format for integration and reasoning, based on triples of the form ⟨subject, predicate, object⟩. A popular language for querying static RDF knowledge bases is SPARQL (Prud'hommeaux and Seaborne, 2008), and C-SPARQL (Barbieri et al., 2009) extends SPARQL for continuous queries, just like CQL extends SQL. A stream is a sequence of timestamped triples, but a query can also return a graph by emitting multiple triples with the same timestamp.

Execution-Model Driven Streaming Languages

Dataflow **synchronous** languages (Benveniste et al., 2003) were introduced in the late 1980s as domain-specific languages for the design of embedded control systems. A dataflow synchronous program executes in a succession of discrete steps, and each step is assumed to be instantaneous (the synchronous hypothesis). A programmer writes high-level specifications in the form of stream functions specifying variable values at each step or instant. Section "Synchronous Dataflow in Lustre" illustrates this approach with the language Lustre (Caspi et al., 1987).

Streaming **big data** is motivated by the "4 Vs": a lot of data (volume) streams quickly (velocity) into the system, which must deal with diverse data and functionality (variety) and with uncertainty (veracity). Languages for big-data streaming let users specify an explicit stream graph that can be easily distributed with minimal synchronization and are extensible by operators in widely adopted general-purpose languages. Section "Big-Data Streaming in SPL" elaborates on this for the concrete example of SPL (Hirzel et al., 2017).

Target-User Driven Streaming Languages

Complex event processing, or **CEP**, lets users compose events hierarchically to span the gap between low-level and high-level concepts. There are various pattern languages for CEP that compile to finite automatons. Recognizing this, the MATCH-RECOGNIZE SQL extension proposal simply adopts familiar regular expressions as the CEP pattern language (Zemke et al., 2007). While the SQL basis focuses on a relational model, regular expressions can also be used for CEP in big-data streaming (Hirzel, 2012).

Since there are many more **spreadsheet** users than software developers, a spreadsheet-based streaming language could reach more target users. Furthermore, spreadsheets are reactive: changes trigger updates to dependent formulas. ActiveSheets hooks up some spreadsheet cells to input or output streams, with normal spreadsheet formulas in between (Vaziri et al., 2014). When the two-dimensional spreadsheet data model is too limiting, it can be augmented with windows and partitioning (Hirzel et al., 2016).

A streaming language based on **natural language** might reach the maximum number of target users. However, since natural language is ambiguous, a controlled natural language (CNL) is a better choice (Kuhn, 2014). For instance, the language for META is a CNL for specifying event-condition-action rules, temporal predicates, and data types (Arnold et al., 2016). The data model includes events and entities with nested concepts and can be shown to be equivalent to the nested relational model (Shinnar et al., 2015).

Examples

This section gives details and concrete code examples for two out of the eight approaches for languages surveyed in the previous section:

the synchronous dataflow approach exemplified by Lustre (Caspi et al., 1987) and the big-data streaming approach exemplified by SPL (Hirzel et al., 2017).

When it comes to implementing streaming languages, there is a spectrum from basic to sophisticated techniques. At the basic end of the spectrum are configuration files in some existing markup format such as XML. The streaming engine interprets the configuration file to construct and then execute a stream graph. An intermediate point is a domain-specific embedded language (EDSL or sometimes DSEL) (Hudak, 1998). As the name implies, an EDSL is a domain-specific language (DSL) that is embedded in some host language, typically a general-purpose language (GPL). The line between simple libraries and EDSLs is blurred, but in general, EDSLs encourage a more idiomatic programming style. Recently, EDSLs have gained popularity as several GPLs have added features that make them more suitable for hosting EDSLs. At the sophisticated end of the spectrum are full-fledged, stand-alone DSLs with their own syntax, compiler, and other tools. While stand-alone streaming DSLs are not embedded in a GPL, they often interface with a GPL, e.g., for user-defined operators.

For clarity of exposition, the following examples use stand-alone streaming languages. Stand-alone languages are the norm for synchronous dataflow, because self-contained code is easier to reason about. On the other hand, for big-data streaming, EDSLs that specify an explicit stream graph are popular, because they are easier to implement. But as the example below illustrates, once implemented, stand-alone languages also have advantages for big-data streaming.

Synchronous Dataflow in Lustre

Synchronous dataflow languages were introduced to ease the design and certification of embedded systems by providing a well-defined mathematical framework that combines a logical notion of time and deterministic concurrency. It is then possible to formally reason about the system, simulate it, prove safety properties, and generate embedded code. The synchronous dataflow language Lustre is the backbone of the industrial language and compiler Scade (Colaco et al., 2017) routinely used to program embedded controllers in many critical applications.

In Lustre a program is a set of equations defining streams of values. Time proceeds by discrete logical steps, and at each step, the program computes the value of each stream depending on its inputs and possibly previously computed values. Consider the example of Fig. 2 adapted from Bourke et al. (2017). The function *counter* takes three input streams, two integer streams *init* and *incr*, and one boolean stream *reset*. It returns the cumulative sum of the values of *incr* initialized with *init* and similarly reset when *reset* is *true* (Line 5). The variable *pn* (Line 4) stores the value of the counter *n* at the previous step using the initialization operator (−>) and the non-initialized delay **pre**.

A stream is not necessarily defined at each step. The clock of a stream is a boolean sequence giving the instants where it is defined. Streams with different clocks can be combined via sampling (**when**) or stuttering (**current**). For instance, the *tracker* function of Fig. 2 tracks the number of times the speed of a vehicle exceeds the speed limit. The **when** operator samples a stream according to a boolean condition. The function *counter* is thus only activated when *x* is *true* (Line 12). The **current** operator completes a stream with the last defined value when it is not present (Line 13). The value of *t* is thus sustained when *x* is *false*. The execution of such a program can be represented as a timeline, called a chronogram (illustrated in Fig. 2), showing the sequence of values taken by its streams at each step.

Specific compilation techniques for synchronous languages exist to generate efficient and reliable code for embedded controllers. Compilers produce imperative code that can be executed in a control loop triggered by external events or on a periodic signal (e.g., every millisecond). The link between logical and real time is left to the designer of the system.

Since the seminal dataflow languages Lustre (Caspi et al., 1987) and Signal (Le Guernic et al., 1991), multiple extensions of the dataflow synchronous model were proposed. Lucid Synchrone (Pouzet, 2006) combines the dataflow

```
1  node counter (init, incr: int; reset: bool) returns (n: int);
2  var pn: int;
3  let
4    pn = init -> pre n;
5    n = if reset then init else pn + incr;
6  tel
7
8  node tracker (speed, limit: int) returns (t: int);
9  var x: bool; cpt: int when x;
10 let
11   x = (speed > limit);
12   cpt = counter((0, 1, false) when x);
13   t = current(cpt);
14 tel
```

speed	28	29	32	30	44	53	58	48	33	28	29	...
limit	30	30	30	30	55	55	55	30	30	30	30	...
x	F	F	T	F	F	F	T	T	T	F	F	...
cpt			1				2	3	4			...
t	0	0	1	1	1	1	2	3	4	4	4	...

Stream Processing Languages and Abstractions, Fig. 2 Lustre code example with a possible execution

synchronous approach with functional features à la ML, the n-synchronous model (Mandel et al., 2010) relaxes the synchronous hypothesis by allowing communication with bounded buffers, and Zélus (Bourke and Pouzet, 2013) is a Lustre-like language extended with ordinary differential equations to define continuous-time dynamics.

Recent efforts focus on the compilation, verification, and test of dataflow synchronous programs. New techniques have been proposed to compile Lustre programs for many-core systems (Rihani et al., 2016) or improve the computation of the worst-case execution time (WCET) of the compiled code (Bonenfant et al., 2017; Forget et al., 2017). Kind2 (Champion et al., 2016) is a verification tool based on SMT solvers to model-check Lustre programs, and the Vélus compiler (Bourke et al., 2017) tackles the problem of verifying the compiler itself using a proof assistant. Lutin (Raymond and Jahier, 2013) (and its industrial counterpart, the Argosim Stimulus tool Argosim, 2015) is a DSL to design non-deterministic test scenarios for Lustre programs.

Big-Data Streaming in SPL

Big-data streaming languages are designed to handle high-throughput streams while at the same time being expressive enough to handle diverse data formats and streaming operators. A popular way to address the requirement of high throughput is to make it easy to execute the streaming application not just on a single core or even a single computer, but on a cluster of computers. And a popular way to address the requirement of high expressiveness is to make it easy for programmers to define new streaming operators, possibly using a different programming language than the stream processing language they use for composing operators into a graph.

SPL is a big-data streaming language designed for distribution and extensibility (Hirzel et al., 2017). It was invented in 2009 and is being actively used in industry (IBM, 2008). An SPL program is an explicit stream graph of streams and operators. Unlike dataflow synchronous languages, and like other big-data streaming languages, SPL uses only minimal synchronization: an operator can fire whenever there is data available on any of its input ports, following semantics formalized in the Brooklet calculus (Soulé et al., 2010). Since synchronization across different cores and computers is hard to do efficiently, reducing synchronization simplifies distribution, giving the runtime system more flexibility for which operators to co-locate in the same core or computer. There is no assumption of simultaneity between different operator firings. When downstream operators cannot keep up with the data

```
1 stream<float64 len, rstring caller> Calls = CallsSource() { }
2
3 stream<float64 len, int32 num, rstring who> CallStats = Aggregate(Calls) {
4     window Calls:    sliding, time(24.0 * 60.0 * 60.0), time(60.0);
5     output CallStats: len = Max(Calls.len),
6                       num = MaxCount(Calls.len),
7                       who = ArgMax(Calls.len, Calls.caller);
8 }
```

Stream Processing Languages and Abstractions, Fig. 3 SPL code example

rate, they implicitly throttle upstream operators via back-pressure.

Figure 3 shows an example SPL program. Line 1 defines a stream *Calls* as the output of invoking an operator *CallsSource*. In SPL, streams carry tuples that are strongly and statically typed and whose fields can hold simple numbers or strings as in the example but can also hold nested lists and tuples. The *CallsSource* operator has no further configuration (empty curly braces); for this example, assume it is user-defined elsewhere. Programmers can define their own operators either in SPL or in other languages such as C++ or Java. Lines 3–8 define a stream *CallStats* by invoking an operator *Aggregate*. The code configures the operator with an input stream *Calls*, with a **window** clause for a 24-h sliding window with 1-minute granularity, and with an **output** clause. While many operators support these and other clauses, they can also contain code restricted to the operator at hand. The *Aggregate* operator in SPL's standard library supports intrinsic functions for *Max*, *MaxCount*, and various other aggregations. Programmers can extend SPL with new operators that, like *Aggregate*, support various configurations and operator-specific intrinsic functions.

SPL was influenced by earlier big-data streaming systems such as Borealis (Abadi et al., 2005) and TelegraphCQ (Chandrasekaran et al., 2003), generalizing them to be less dependent on relational data and more extensible. Various other streaming systems after SPL, such as Storm (Toshniwal et al., 2014) and Spark Streaming (Zaharia et al., 2013), also target big-data streaming. Like SPL, they use explicit stream graphs as their core abstraction, but unlike SPL, they use embedded (not stand-alone) domain-specific languages.

While the examples of Lustre and SPL draw a stark contrast between synchronous dataflow and big-data streaming, there are also commonalities. For instance, the StreamIt language is synchronous, but like big-data streaming languages, it uses an explicit stream graph as its core abstraction (Thies et al., 2002). And Soulé et al. show how to reduce the dependence of StreamIt on synchrony (Soulé et al., 2013).

Future Directions for Research

The landscape of streaming languages is far from consolidating on any dominant approach. New languages keep coming out to address a variety of open issues. One active area of research is the interaction between streams (data in motion) and state (data at rest). While CQL gave a conceptually clean answer (Arasu et al., 2006), people are debating alternative approaches, such as the Lambda architecture (Marz, 2011) and the Kappa architecture (Kreps, 2014). Another active area of research is how to handle uncertainty, such as out-of-order data, missing fields, erroneous sensor readings, approximate algorithms, or faults. On this front, streaming languages have not yet reached the clarity of databases with their ACID properties. When it comes to implementation strategies, there has been a recent surge in embedded domain-specific languages. But while EDSLs have fewer tooling needs and are less intimidating for users familiar with their host language, they are less self-contained and offer less static optimization and error-checking than stand-alone languages. We hope this article inspires innovation in streaming languages that are well-informed by those that came before.

Cross-References

- Continuous Queries
- Languages for Big Data Analysis

References

Abadi DJ, Ahmad Y, Balazinska M, Cetintemel U, Cherniack M, Hwang JH, Lindner W, Maskey AS, Rasin A, Ryvkina E, Tatbul N, Xing Y, Zdonik S (2005) The design of the Borealis stream processing engine. In: Conference on innovative data systems research (CIDR), pp 277–289

Arasu A, Babu S, Widom J (2006) The CQL continuous query language: semantic foundations and query execution. J Very Large Data Bases (VLDB J) 15(2):121–142

Argosim (2015) Stimulus. http://argosim.com/. Retrieved Nov 2017

Arnold M, Grove D, Herta B, Hind M, Hirzel M, Iyengar A, Mandel L, Saraswat V, Shinnar A, Siméon J, Takeuchi M, Tardieu O, Zhang W (2016) META: middleware for events, transactions, and analytics. IBM J Res Dev 60(2–3):15:1–15:10

Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Symposium on principles of database systems (PODS), pp 1–16

Barbieri DF, Braga D, Ceri S, Della Valle E, Grossniklaus M (2009) C-SPARQL: SPARQL for continuous querying. In: Poster at international World Wide Web conferences (WWW-poster), pp 1061–1062

Benveniste A, Caspi P, Edwards SA, Halbwachs N, Le Guernic P, De Simone R (2003) The synchronous languages 12 years later. Proc IEEE 91(1):64–83

Bonenfant A, Carrier F, Cassé H, Cuenot P, Claraz D, Halbwachs N, Li H, Maiza C, De Michiel M, Mussot V, Parent-Vigouroux C, Puaut I, Raymond P, Rohou E, Sotin P (2016) When the worst-case execution time estimation gains from the application semantics. In: European congress on embedded real-time software and systems (ERTS2)

Bourke T, Pouzet M (2013) Zélus: a synchronous language with odes. In: Conference on hybrid systems: computation and control (HSCC), pp 113–118

Bourke T, Brun L, Dagand P, Leroy X, Pouzet M, Rieg L (2017) A formally verified compiler for Lustre. In: Conference on programming language design and implementation (PLDI), pp 586–601

Caspi P, Pilaud D, Halbwachs N, Plaice JA (1987) LUSTRE: a declarative language for real-time programming. In: Symposium on principles of programming languages (POPL), pp 178–188

Champion A, Mebsout A, Sticksel C, Tinelli C (2016) The Kind 2 model checker. In: Conference on computer aided verification (CAV), pp 510–517

Chandrasekaran S, Cooper O, Deshpande A, Franklin MJ, Hellerstein JM, Hong W, Krishnamurthy S, Madden S, Raman V, Reiss F, Shah MA (2003) TelegraphCQ: continuous dataflow processing for an uncertain world. In: Conference on innovative data systems research (CIDR)

Chen J, DeWitt DJ, Tian F, Wang Y (2000) NiagaraCQ: a scalable continuous query system for internet databases. In: International conference on management of data (SIGMOD), pp 379–390

Colaco JL, Pagano B, Pouzet M (2017) Scade 6: a formal language for embedded critical software development. In: International symposium on theoretical aspect of software engineering (TASE), pp 1–10

Diao Y, Fischer PM, Franklin MJ, To R (2002) YFilter: efficient and scalable filtering of XML documents. In: Demo at international conference on data engineering (ICDE-Demo), pp 341–342

Forget J, Boniol F, Pagetti C (2017) Verifying end-to-end real-time constraints on multi-periodic models. In: International conference on emerging technologies and factory automation (ETFA)

Hirzel M (2012) Partition and compose: parallel complex event processing. In: Conference on distributed event-based systems (DEBS), pp 191–200

Hirzel M, Rabbah R, Suter P, Tardieu O, Vaziri M (2016) Spreadsheets for stream processing with unbounded windows and partitions. In: Conference on distributed event-based systems (DEBS), pp 49–60

Hirzel M, Schneider S, Gedik B (2017) SPL: an extensible language for distributed stream processing. Trans Program Lang Syst (TOPLAS) 39(1):5:1–5:39

Hudak P (1998) Modular domain specific languages and tools. In: International conference on software reuse (ICSR), pp 134–142

IBM (2008) IBM streams. https://ibmstreams.github.io. Retrieved Nov 2017

Jain N, Mishra S, Srinivasan A, Gehrke J, Widom J, Balakrishnan H, Cetintemel U, Cherniack M, Tibbets R, Zdonik S (2008) Towards a streaming SQL standard. In: Conference on very large data bases (VLDB), pp 1379–1390

Johnston WM, Hanna JRP, Millar RJ (2004) Advances in dataflow programming languages. ACM Comput Surv (CSUR) 36(1):1–34

Kreps J (2014) Questioning the Lambda architecture. http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html. Retrieved Nov 2017

Kuhn T (2014) A survey and classification of controlled natural languages. Comput Linguist 40(1):121–170

Le Guernic P, Gautier T, Le Borgne M, Le Maire C (1991) Programming real-time applications with SIGNAL. Proc IEEE 79(9):1321–1336

Mandel L, Plateau F, Pouzet M (2010) Lucy-n: a n-synchronous extension of Lustre. In: International conference on mathematics of program construction (MPC), pp 288–309

Marz N (2011) How to beat the CAP theorem. http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html. Retrieved Nov 2017

Mendell M, Nasgaard H, Bouillet E, Hirzel M, Gedik B (2012) Extending a general-purpose streaming system

for XML. In: Conference on extending database technology (EDBT), pp 534–539
Pouzet M (2006) Lucid synchrone, version 3, Tutorial and reference manual
Prud'hommeaux E, Seaborne A (2008) SPARQL query language for RDF. W3C recommendation. http://www.w3.org/TR/rdf-sparql-query/. Retrieved Nov 2017
Raymond P, Jahier E (2013) Lutin reference manual version Trilby-1.54
Rihani H, Moy M, Maiza C, Davis RI, Altmeyer S (2016) Response time analysis of synchronous data flow programs on a many-core processor. In: Conference on real-time networks and systems (RTNS), pp 67–76
Shinnar A, Siméon J, Hirzel M (2015) A pattern calculus for rule languages: expressiveness, compilation, and mechanization. In: European conference on object-oriented programming (ECOOP), pp 542–567
Soulé R, Hirzel M, Grimm R, Gedik B, Andrade H, Kumar V, Wu KL (2010) A universal calculus for stream processing languages. In: European symposium on programming (ESOP), pp 507–528
Soulé R, Gordon MI, Amarasinghe S, Grimm R, Hirzel M (2013) Dynamic expressivity with static optimization for streaming languages. In: Conference on distributed event-based systems (DEBS), pp 159–170
Soulé R, Hirzel M, Gedik B, Grimm R (2016) River: an intermediate language for stream processing. Softw Pract Exp (SP&E) 46(7):891–929
Stephens R (1997) A survey of stream processing. Acta Informatica 34(7):491–541
Thies W, Karczmarek M, Gordon M, Maze D, Wong J, Hoffmann H, Brown M, Amarasinghe S (2002) StreamIt: a compiler for streaming applications. Technical report LCS-TM-622, MIT
Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J, Bhagat N, Mittal S, Ryaboy D (2014) Storm @Twitter. In: International conference on management of data (SIGMOD), pp 147–156
Vaziri M, Tardieu O, Rabbah R, Suter P, Hirzel M (2014) Stream processing with a spreadsheet. In: European conference on object-oriented programming (ECOOP), pp 360–384
Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I (2013) Discretized streams: fault-tolerant streaming computation at scale. In: Symposium on operating systems principles (SOSP), pp 423–438
Zemke F, Witkowski A, Cherniak M, Colby L (2007) Pattern matching in sequences of rows. Technical report, ANSI Standard Proposal

Stream Processing Optimization

▸ Stream Query Optimization

Stream Query Optimization

Martin Hirzel[1], Robert Soulé[2], Buğra Gedik[3], and Scott Schneider[1]
[1]IBM Research AI, Yorktown Heights, NY, USA
[2]Università della Svizzera Italiana (USI), Lugano, Switzerland
[3]Department of Computer Engineering, Bilkent University, Ankara, Turkey

Synonyms

Continuous query optimization; Stream processing optimization

Definitions

Stream query optimization is the process of modifying a stream processing query, often by changing its graph topology and/or operators, with the aim of achieving better performance (such as higher throughput, lower latency, or reduced resource usage), while preserving the semantics of the original query.

Overview

A stream query optimization modifies a stream query to make it faster. Users want stream queries to be fast for several reasons. They want to grasp opportunities or avert risks observable on the input streams before it is too late. They want any views derived from the input streams to be up-to-date and not stale. And they want their system to keep up with the rate of input streams without falling behind, which would require shedding load or saving data to disk for later processing.

Knowing about stream query optimizations helps developers at all layers. Application developers who know about stream query optimizations can get the most out of the optimizations built into their streaming platform and can supplement them by hand-optimizing their application where necessary. Streaming platform developers can use knowledge about

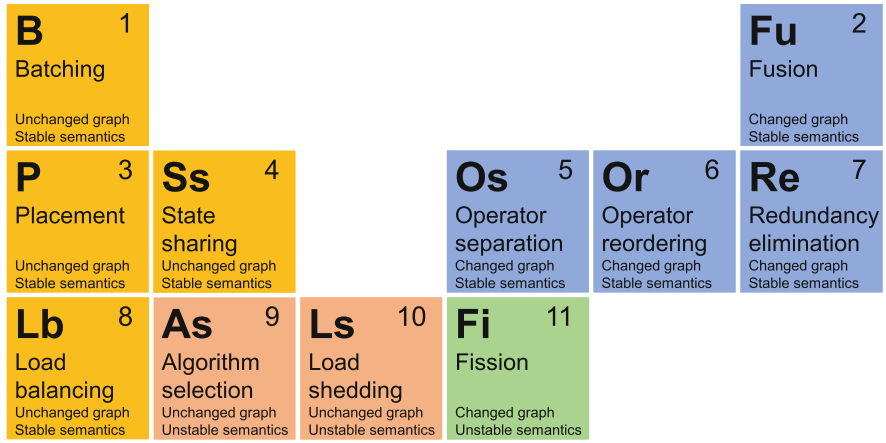
stream query optimizations to make their platform faster by implementing additional optimizations or by generalizing their existing optimizations to apply in more situations. Finally, researchers who invent new optimizations need to know the state-of-the-art optimizations to channel their efforts into the most innovative and impactful direction.

The rest of this section introduces some basic concepts and gives a high-level overview and categorization of the most common stream query optimizations.

An optimization should be both safe and profitable. An optimization is *safe* if it can be applied to a stream query without changing what it computes, as determined by the user's requirements. An optimization is *profitable* if it makes the stream query faster, as measured by metrics that matter to the user, such as throughput, latency, or resource efficiency. There is a substantial literature on different stream query optimizations, with different safety and profitability characteristics. This entry lists the most common optimizations along with short descriptions. More in-depth descriptions can be found in our survey paper and tutorial on stream processing optimizations (Hirzel et al., 2014; Schneider et al., 2013).

Stream query optimizations are best understood with respect to stream graphs. A *stream graph* is a directed graph whose edges are streams and whose nodes are operators. Root and leaf nodes are called sources and sinks, respectively. This entry uses terminology that makes only few assumptions so as not to unnecessarily restrict its scope. For instance, this entry does not assume restrictions on the shape of the stream graph: unless specified otherwise, it does not assume that stream graphs are acyclic, or are single-source-single-sink, or are trees. A *stream* is an ordered sequence of data items, which are values that can range from simple numbers to flat tuples to more elaborate structured data that may be deeply nested and have variable size. Streams are conceptually infinite, in the sense that as the streaming computation unfolds over time, the sequence of data items is unbounded in length. *Operators* are primarily stream transformers but can also have state and side effects beyond the output streams they produce. Indeed, sources and sinks are operators that typically have the side effect of continuously consuming input from and producing output to the external world outside of the stream graph.

Figure 1 lists the most popular stream query optimizations. Each optimization has a symbol (e.g., **B**), a name (e.g., Batching), and two annotations indicating its effect on the graph and the semantics. The optimizations to the left (shown in orange and red) leave the stream graph unchanged. This means the graph still contains the same nodes and edges, and any changes are limited to the behavior of individual operators. The optimizations on the right (shown in blue and green) change the stream graph. The optimizations on the top (shown in orange and blue)



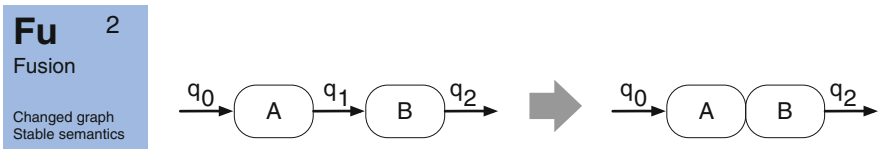
Stream Query Optimization, Fig. 1 Overview of stream query optimizations discussed in this entry

keep the semantics stable. This means that as long as their safety preconditions are satisfied, the observable behavior of the stream query is the same before and after applying the optimization. These optimizations are safe in the sense discussed above. The extreme case is batching and fusion at the very top, which are not only safe, but have safety preconditions that are trivial to satisfy. Only the threeoptimizations at the very bottom have unstable semantics: load shedding, which always changes the semantics, and algorithm selection and fission, which sometimes change semantics if the algorithm is approximate or if the fission perturbs the order of data items. These forms of semantic changes are sometimes tolerable depending on application requirements.

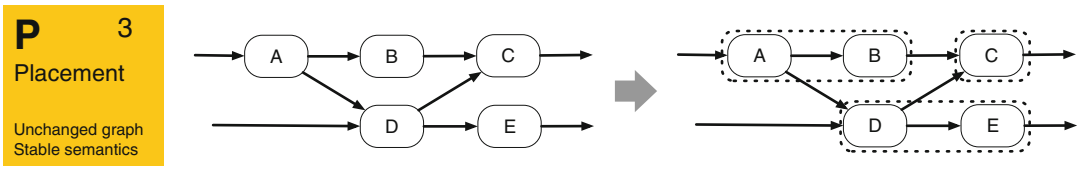
The following section will elaborate on each of the optimizations from Fig. 1 with illustrations, definitions, and literature references.

Key Research Findings

Batching *Definition:* Batching reduces overhead by processing multiple data items together. This optimization improves throughput by amortizing the cost of operator-firing and communication over several data items. However, this throughput gain is usually at the expense of additional latency, as an operator cannot fire until it has received a batch-size number of data items. *References:* In the literature, batching is also called train scheduling (Carney et al., 2003) and execution scaling (Gordon et al., 2006). Batching can be dynamic, as in SEDA (Welsh et al., 2001), or static, as in StreamIt (Gordon et al., 2006).

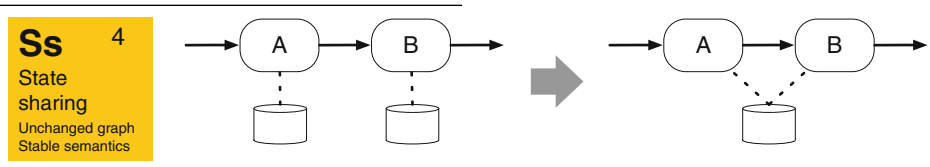


Fusion *Definition:* Fusion combines smaller operators into a larger one, to avoid the overhead of decreased pipeline parallelism. *References:* StreamIt (Gordon et al., 2002) aggressively fuses fine-grained operators, followed by fission. In Aurora, fusion is referred to as superbox scheduling (Tatbul et al., 2003). System S uses the COLA fusion optimizer (Khandekar et al., 2009), which balances safety and profitability constraints.

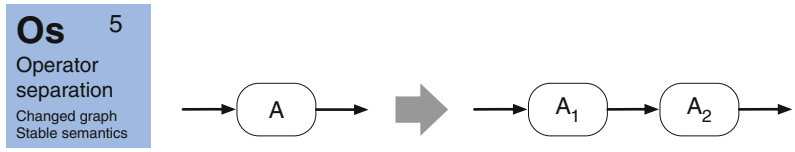


Placement *Definition:* Placement assigns operators to hosts and cores to reduce communication costs or better utilize available resources. Frequently, these two goals are at odds. When multiple operators are placed on the same host, they communicate at lower cost, but they compete for common resources, such as disk, memory, or CPU. On the other hand, when operators

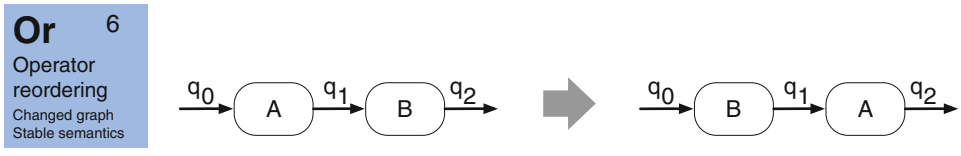
are placed on different hosts, that reduces contention but incurs higher communication costs. *References:* StreamIt uses placement to optimize streaming applications deployed on multi-core machines with nonuniform memory access (Gordon et al., 2002). Pietzuch et al. use metrics gathered from network conditions to place operators in a distributed setting (Pietzuch et al., 2006). SODA incorporates job admission with the placement decisions (Wolf et al., 2008).



State sharing *Definition:* State sharing attempts to avoid unnecessary copies of data. While the main goal of the optimization is to reduce the memory footprint of a streaming application, it can also impact performance by reducing stalls due to cache misses or disk I/O. *References:* State sharing can be applied generally between streaming operators, such as in the work of Brito et al. (2008). Or, it can be applied in more restricted forms, such as in CQL, which shares only window state (Arasu et al., 2006), or in the work of Sermulins et al., which shares queue state between two pipelined operators (Sermulins et al., 2005).

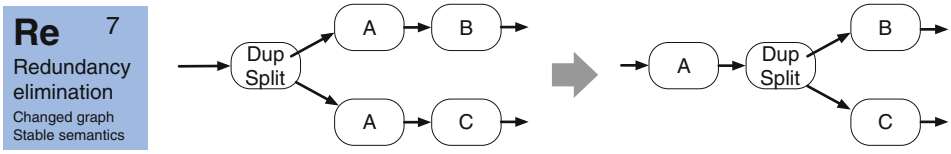


Operator separation *Definition:* Operator separation splits a large computation into smaller steps. In some cases, this optimization can result in reduced resource consumption. Often, this optimization is used to enable other optimizations, such as operator reordering. *References:* Using algebraic equivalences for separating operators is a common technique in database query execution planning (Garcia-Molina et al., 2008). Yu et al. (2009) present a stream query compiler that uses explicit annotations to determine when to separate aggregate operators. Decoupled software pipelining separates general code by analyzing data dependencies from first principles (Ottoni et al., 2005).

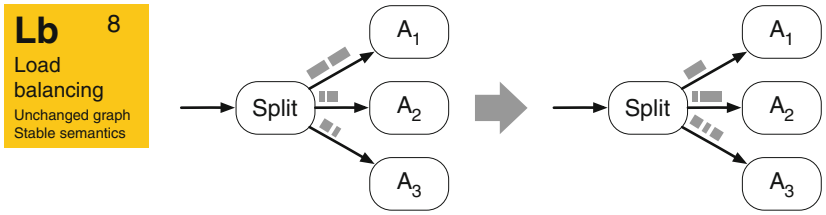


Operator reordering *Definition:* A reordering optimization moves more selective operators, which reduce the data volume, upstream. This has the benefit of reducing the data flowing into downstream computation, thus eliminating unnecessary work. However, care must be taken to preserve the desired semantics, and operators should only be re-ordered if the operations are commutative. *References:* Reordering based on the properties of relational algebra is common in database query planning (Garcia-Molina et al., 2008). The Volcano (Graefe, 1990) system

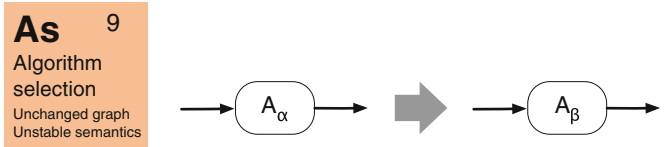
implements a particularly profitable case of reordering: swapping split and merge operators in a data-parallel pipeline to avoid choke-points. Eddies (Avnur et al., 2000) is a dynamic technique for finding the most profitable ordering of operators with independent selectivities.



Redundancy elimination *Definition:* Redundancy elimination eliminates superfluous computations. This optimization must ensure that removing a given computation does not change the resulting output. While, in general, determining program equivalence is undecidable, in practice, streaming languages are often based on an algebra, which makes the optimization feasible. *References:* The Rete algorithm (Forgy, 1982) is a highly-influential approach to detecting and eliminating redundancies in a multi-tenant system. NiagaraCQ (Chen et al., 2000) applied similar ideas to processing streaming XML. Pietzuch et al. (2006) eliminate redundancy at application launch time.

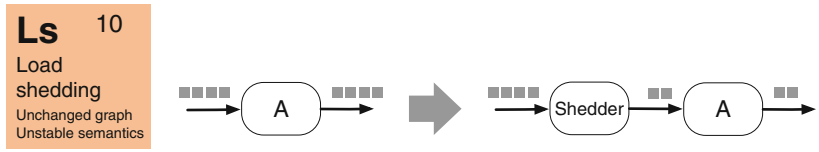


Load balancing *Definition:* Load balancing attempts to distribute workload evenly across resources. To be effective, a load-balancing optimization must adapt to workload skew, e.g., when there are many accesses to a popular data item. *References:* River uses intelligent routing for load balancing in a cluster (Arpaci-Dusseau et al., 1999). Caneill et al. use online adaptive routing, which considers downstream communication (Caneill et al., 2016). In contrast, Amini et al. use operator placement for load balancing in a cluster (Amini et al., 2006). StreamIt also uses placement for load balancing, but targets multi-core machines (Gordon et al., 2002).

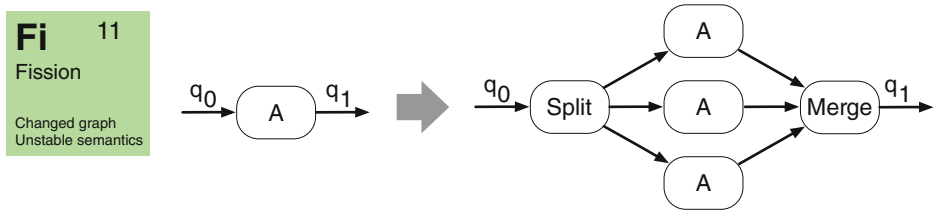


Algorithm selection *Definition:* Algorithm selection uses a different algorithm to implement an operator. There are various reasons to change the algorithm. For example, one algorithm may be more general than another. One algorithm may optimize for different criteria. Or, one algorithm may perform better than others under different circumstances. *References:* Changing the operator algorithm for particular workloads is common in database systems (e.g., using a hash join vs.

a nested loop join Garcia-Molina et al., 2008). In streaming systems, SEDA allows an operator to pick a different algorithm to provide degraded service (Welsh et al., 2001). Borealis allows an operator to switch to a different algorithm based on a control input (Abadi et al., 2005). And, SODA offers algorithm selection at the granularity of entire jobs (Wolf et al., 2008).



Load shedding *Definition:* Load shedding copes with high load by dropping data items to process. Load shedding may change the expected results of a computation, and therefore the semantics of the streaming application. *References:* Aurora implements priority-based load shedding (Tatbul et al., 2003). Compact Shedding Filters perform load-shedding at data-generating sensors, rather than the server, to avoid unnecessary network communication (Gedik et al., 2008).



Fission *Definition:* Fission, often referred to as data parallelism, attempts to process multiple data items in parallel by replicating an operator. Note that when parallelizing operators, the optimizer must respect ordering or state constraints to ensure the correctness of the computation. *References:* The StreamIt compiler applies fission to stateless operators with static selectivity (Gordon et al., 2006). Schneider et al. explored how to make fission safe in the more general case of partitioned-stateful and selective operators (Schneider et al., 2015). Finally, Brito et al. propose using transactional memory to make fission safe in the case of arbitrary operator state (Brito et al., 2008).

Examples of Application

We illustrate the use of the three most popular optimizations, namely, fission, fusion, and batching, in the context of real-world streaming applications from the literature.

Fission One application that benefits a lot from fission is streaming radio astronomy, which forms evolving imaging maps of radio emission from the sky over a wide range of frequencies (Biem et al., 2010b). High-performance processing is a critical requirement in this application, due to the sheer volume of sensor data that flow in real-time from a very large array of antennas, which is typical of phased array radio telescopes (see SKA 2000). The application's flow graph is organized as a split-merge topology. An initial operator performs frequency mapping and blocking so that a subsequent operator splits the data into blocks of frequency channels. Each such block is then processed in parallel by performing indexing and convolution operations, forming a costly stateless parallel region that highly benefits from fission. Finally, the results are merged via an aggregation to form the complete imaging map.

Fusion In Biem et al. (2010a), a streaming application is presented for maintaining live traffic

status information using GPS sensor data. The application processes floating car data originating from public transportation vehicles to extract up-to-date traffic information, such as speed and traffic flow measurements at the level of streets within a city, traffic volume measurements by region, estimates of travel times between different points, etc. The application is organized as a graph of streams and operators, where different operators perform individual tasks, such as data parsing and cleaning, snapping GPS points to roads, aggregation and statistics maintenance, prediction of travel times, etc. The operators in the flow graph of the application are grouped into processing elements, which are then distributed across machines. The operators that are assigned to the same processing element are fused inside a shared address space, in order to reduce the data transfer latency.

Batching The batching optimization is particularly useful when interacting with external systems. A common use case for such interaction is managing state. For instance, LinkedIn, which is a business- and employment-oriented social networking service, runs several streaming applications that manage state, such as user profiles and aggregate counts. These applications include email digest generation, top-k relevant category detection, and profile update standardization, among others (Noghabi et al., 2017). In these applications, the state needs to be accessed from either the local disk-based or remote state management systems. Similarly, many of these streaming applications write their output to an external system, such as a message queue. When an external interaction is to be performed on a per-tuple basis, batching is an effective optimization that can amortize the overheads and significantly improve the performance.

Future Directions for Research

In an ideal world, programmers would code their streaming applications at the most natural level of abstraction without having to worry about runtime performance, and the streaming system would automatically execute the applications with consistently high performance. While stream query optimizations have made significant advances toward this goal, they still fall short on automation and predictability.

It is still difficult to fully automate streaming optimizations, because automatic optimizers may not find the most profitable setting. Finding the most profitable setting ahead-of-time, before executing the streaming application, is tricky because the performance model may be complicated (e.g., when multiple optimizations interact) or some information required by the performance model may be missing or hard to predict. An alternative to ahead-of-time optimization is online feedback-directed optimization, which is subject to the trade-offs inherent in the SASO properties of feedback control (stability, accuracy, settling, and no overshoot) (Hellerstein et al., 2004). Work such as that of De Matteis and Mencagli, which uses a control-theoretic approach to perform online adaptations to optimize for latency (De Matteis et al., 2016), is a promising direction.

When streaming applications fall short of their expected peak performance, programmers often optimize them by hand. This has the advantage of giving programmers more control over squeezing the last bit of performance out of their application. Unfortunately, it can also clutter their code, can make the performance brittle by over-fitting to the current workload and execution environment, and may even inhibit automated optimizations. One approach to address these issues is to decouple the optimizer hints and directives from the core application logic (Hirzel et al., 2017).

Overall, there is still much work to be done in developing more effective optimizations, more reliable performance models, and more unintrusive language features for giving optimizer hints. This entry represents a snapshot of the state of the art, and we encourage the reader to venture beyond it.

Cross-References

- ▸ Continuous Queries
- ▸ Introduction to Stream Processing Algorithms
- ▸ Sliding-Window Aggregation Algorithms

References

Abadi DJ, Ahmad Y, Balazinska M, Çetintemel U, Cherniack M, Hwang JH, Lindner W, Maskey AS, Rasin A, Ryvkina E, Tatbul N, Xing Y, Zdonik S (2005) The design of the Borealis stream processing engine. In: Conference on innovative data systems research (CIDR), pp 277–289

Amini L, Jain N, Sehgal A, Silber J, Verscheure O (2006) Adaptive control of extreme-scale stream processing systems. In: International conference on distributed computing systems (ICDCS)

Arasu A, Babu S, Widom J (2006) The CQL continuous query language: semantic foundations and query execution. J Very Large Data Bases (VLDB J) 15(2): 121–142

Arpaci-Dusseau RH, Anderson E, Treuhaft N, Culler DE, Hellerstein JM, Patterson D, Yelick K (1999) Cluster I/O with river: making the fast case common. In: Workshop on I/O in parallel and distributed systems (IOPADS), pp 10–22

Avnur R, Hellerstein JM (2000) Eddies: continuously adaptive query processing. In: International conference on management of data (SIGMOD), pp 261–272

Biem A, Bouillet E, Feng H, Ranganathan A, Riabov A, Verscheure O, Koutsopoulos HN, Rahmani M, Guc B (2010a) Real-time traffic information management using stream computing. IEEE Data Eng Bull 33(2): 64–68

Biem A, Elmegreen B, Verscheure O, Turaga D, Andrade H, Cornwell T (2010b) A streaming approach to radio astronomy imaging. In: Conference on acoustics, speech, and signal processing (ICASSP), pp 1654–1657

Brito A, Fetzer C, Sturzrehm H, Felber P (2008) Speculative out-of-order event processing with software transaction memory. In: Conference on distributed event-based systems (DEBS), pp 265–275

Caneill M, El Rheddane A, Leroy V, De Palma N (2016) Locality-aware routing in stateful streaming applications. In: International conference on middleware, pp 4:1–4:13

Carney D, Cetintemel U, Rasin A, Zdonik S, Cherniack M, Stonebraker M (2003) Operator scheduling in a data stream manager. In: Conference on very large data bases (VLDB), pp 309–320

Chen J, DeWitt DJ, Tian F, Wang Y (2000) NiagaraCQ: a scalable continuous query system for internet databases. In: International conference on management of data (SIGMOD), pp 379–390

De Matteis T, Mencagli G (2016) Keep calm and react with foresight: strategies for low-latency and energy-efficient elastic data stream processing. In: Principles and practice of parallel programming (PPoPP), pp 13:1–13:12

Forgy CL (1982) Rete: a fast algorithm for the many pattern/many object pattern match problem. Artif Intell 19:17–37

Garcia-Molina H, Ullman JD, Widom J (2008) Database systems: the complete book, 2nd edn. Prentice Hall, Upper Saddle River

Gedik B, Wu KL, Yu PS (2008) Efficient construction of compact shedding filters for data stream processing. In: International conference on data engineering (ICDE), pp 396–405

Gordon MI, Thies W, Karczmarek M, Lin J, Meli AS, Lamb AA, Leger C, Wong J, Hoffmann H, Maze D, Amarasinghe S (2002) A stream compiler for communication-exposed architectures. In: Conference on architectural support for programming languages and operating systems (ASPLOS), pp 291–303

Gordon MI, Thies W, Amarasinghe S (2006) Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. In: Conference on architectural support for programming languages and operating systems (ASPLOS), pp 151–162

Graefe G (1990) Encapsulation of parallelism in the Volcano query processing system. In: International conference on management of data (SIGMOD), pp 102–111

Hellerstein JL, Diao Y, Parekh S, Tilbury DM (2004) Feedback control of computing systems. Wiley, Hoboken

Hirzel M, Soulé R, Schneider S, Gedik B (2014) A catalog of stream processing optimizations. ACM Comput Surv (CSUR) 46(4):1–34

Hirzel M, Schneider S, Gedik B (2017) SPL: an extensible language for distributed stream processing. Trans Program Lang Syst (TOPLAS) 39(1):5: 1–5:39

Khandekar R, Hildrum I, Parekh S, Rajan D, Wolf J, Wu KL, Andrade H, Gedik B (2009) COLA: optimizing stream processing applications via graph partitioning. In: International conference on middleware, pp 308–327

Noghabi SA, Paramasivam K, Pan Y, Ramesh N, Bringhurst J, Gupta I, Campbell RH (2017) Samza: stateful scalable stream processing at LinkedIn. In: Conference on very large data bases (VLDB), pp 1634–1645

Ottoni G, Rangan R, Stoler A, August DI (2005) Automatic thread extraction with decoupled software pipelining. In: International symposium on microarchitecture (MICRO), pp 105–118

Pietzuch P, Ledlie J, Schneidman J, Roussopoulos M, Welsh M, Seltzer M (2006) Network-aware operator placement for stream-processing systems. In: International conference on data engineering (ICDE), pp 49–61

Schneider S, Gedik B, Hirzel M (2013) Tutorial: stream processing optimizations. In: Conference on distributed event-based systems (DEBS), pp 249–258

Schneider S, Hirzel M, Gedik B, Wu KL (2015) Safe data parallelism for general streaming. IEEE Trans Comput (TC) 64(2):504–517

Sermulins J, Thies W, Rabbah R, Amarasinghe S (2005) Cache aware optimization of stream programs. In:

Conference on languages, compiler, and tool support for embedded systems (LCTES), pp 115–126
SKA Telescope (2000) Square kilometre array telescope. https://skatelescope.org. Retrieved Nov 2017
Tatbul N, Cetintemel U, Zdonik S, Cherniack M, Stonebraker M (2003) Load shedding in a data stream manager. In: Conference on very large data bases (VLDB), pp 309–320
Welsh M, Culler D, Brewer E (2001) SEDA An architecture for well-conditioned, scalable Internet services. In: Symposium on operating systems principles (SOSP), pp 230–243
Wolf J, Bansal N, Hildrum K, Parekh S, Rajan D, Wagle R, Wu KL, Fleischer L (2008) SODA: an optimizing scheduler for large-scale stream-based distributed computer systems. In: International conference on middleware, pp 306–325
Yu Y, Gunda PK, Isard M (2009) Distributed aggregation for data-parallel computing: interfaces and implementations. In: Symposium on operating systems principles (SOSP), pp 247–260

Stream Reasoning

▸ Semantic Stream Processing

Stream Reduce

▸ Sliding-Window Aggregation Algorithms

Stream Window Aggregation Semantics and Optimization

Paris Carbone[1], Asterios Katsifodimos[2], and Seif Haridi[1]
[1]KTH Royal Institute of Technology, Stockholm, Sweden
[2]TU Delft, Delft, Netherlands

Definitions

`Sliding windows` are bounded sets which evolve together with an infinite data stream of records. Each new sliding window evicts records from the previous one while introducing newly arrived records as well. Aggregations on windows typically derive some metric such as an average or a sum of a value in each window. The main challenge of applying aggregations to sliding windows is that a naive execution can lead to a high degree of redundant computation due to a large number of common records across different windows. Special optimization techniques have been developed throughout the years to tackle redundancy and make sliding window aggregation feasible and more efficient in large data streams.

Overview

Data stream processing has evolved significantly throughout the years, both in terms of system support and in programming model primitives. Alongside adopting common data-centric operators from relational algebra and functional programming such as select, join, flatmap, reduce, etc., stream processors introduced a new set of primitives that are exclusive to the evolving nature of unbounded data. Stream windows are perhaps the most common and widely studied primitive in stream processing which is used to express computation on continuously evolving subsets out of a possibly never-ending stream. In essence, stream windows grant control on the granularity and the scope of stream aggregations.

Several early stream processing systems (e.g., TelegraphCQ (Chandrasekaran et al., 2003), STREAM (Arasu et al., 2004)) provided support for windowing through a predefined set of primitives to construct time- and count-based sliding windows. For example, periodic tumbling and sliding windows were already standardized as early as the SQL-99 standard and studied thoroughly in the Continuous Query Language (CQL) (Arasu et al., 2006) as well as the stream processing language (SPL) (Hirzel et al., 2009) among others. A *tumbling* window is a simple case of a stream window type, which is defined as a sequence of periodic consecutive sets of records in a stream with a fixed length that is termed *range*. For example, if we assume a stream of car speed events, the following simple query in CQL would discretize that stream into

windows of every 30sec interval and compute the maximum speed per window:

```
SELECT max(speed)
from CarEvents
[RANGE 30 Seconds]
```

In principle, in tumbling windows each record can only belong to a single window. As a result, the evaluation of each window can be performed trivially by grouping records by the window they belong to and executing each window aggregation independently. However, sliding windows add a challenging twist to the formula, namely, the "slide." As an example consider the following sliding window query in SQL-99:

```
SELECT AVERAGE(speed)
FROM CarEvents
[WATTR timestamp
 RANGE 7 minute
 SLIDE 3 minute]
```

The *slide* represents "when" or "how often" a window has to be evaluated while including all records defined in its *range* in the computation of the average speed. In this sliding window example, there is an overlap of 5 min between each consecutive window, and thus, a naive execution would result into redundant operations in its great majority. Beyond periodic windows, today's Apache open-source systems such as Flink (Carbone et al., 2015, 2017), Beam, and Apex provide support for more advanced, often user-defined, sliding window definitions such as session (Akidau et al., 2015) or content-driven windows (Bifet and Gavalda, 2007), among others.

Sliding windows have their own set of optimization techniques that aim to reduce the redundant computations caused by the intersection of events between neighboring windows. In this paper we categorize optimization techniques into *slicing*, *pre-aggregation*, and *hybrid* and analyze them throughout the rest of this chapter.

Basic Concepts

There are many interpretations of window semantics, from simple range and count event-time windows (Arasu et al., 2006) to policy-based (Hirzel et al., 2009) and composite event-time windows with retraction for out-of-order processing (Li et al., 2008b). The SECRET model (Botan et al., 2010) aimed to subsume most of the windowing semantics proposed in academia and commercial systems. However, for the sake of brevity, a more simplified description is used here, with a heavy focus on window aggregation, based on the recent work on the Cutty aggregator (Carbone et al., 2016) and FlatFat (Tangwongsan et al., 2015).

Stream Discretization

Data streams are unbounded sequences of records which are described by a given schema T. More formally, a stream $\overline{s} \in \text{Seq}(T)$ is a sequence out of all possible sequences $\text{Seq}(T)$ over T. Windows are finite subsequences reflecting intervals of a stream $\overline{s}$. An interval $s[a, b]$ is simply a set of records from index a to b over a stream $\overline{s}$ and the set of all possible intervals $\text{Str}(T) \subset \text{Seq}(T)$.

In their most general form, windows can be derived by *discretizing* an unbounded stream. A `Discretizer` transforms a stream $\overline{s} \in \text{Seq}(T)$ into a sequence $\overline{w} \in \text{Seq}(\text{Str}(T))$ of (possibly overlapping) windows. In the most system-agnostic manner, every possible discretization of a stream can be aided through a discretization function provided to a special *Discretize* or *Windowing* stream operator.

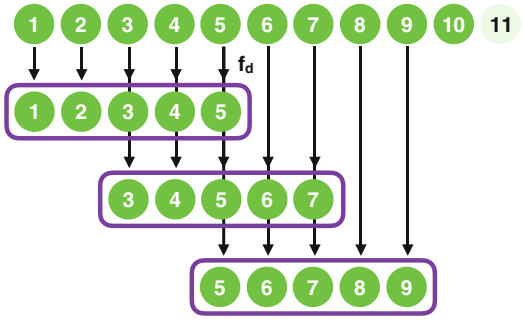
Definition 1 $\texttt{Discretize} : f_{\text{disc}} \times \text{Seq}(T) \rightarrow \text{Seq}(\text{Str}(T))$

Figure 1 depicts a simple example of a discretization function f_d applied on a stream of elements which forms count windows with a "range" of 5 records and a "slide" of 2 records.

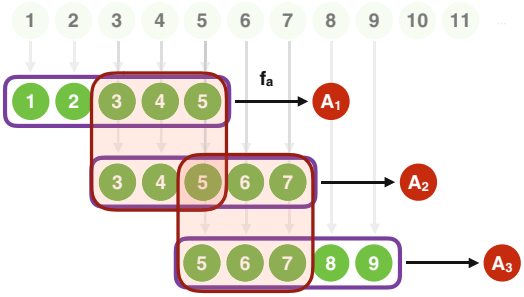
Aggregating Sliding Windows

Conceptually, in data stream programming models, an `Aggregate` operator computes an aggregation on each window derived after a stream discretization, given an aggregation function f_a.

Definition 2 $\texttt{Aggregate} : (f_a : \text{Str}(T) \rightarrow T') \times \text{Seq}(\text{Str}(T)) \rightarrow \text{Seq}(T')$



Stream Window Aggregation Semantics and Optimization, Fig. 1 Discretization of a count window of range 5 and slide 2



Stream Window Aggregation Semantics and Optimization, Fig. 2 Aggregation of a count window of range 5 and slide 2

Examples of f_a is a `SUM` or `AVG`, but also more complex aggregations can be executed in a window, such as building a machine learning model. Figure 2 shows how aggregates are formed on each consecutive window of a discretized stream. The main reason for optimizing the window aggregation process stems from two issues that can be observed in this example. First, a consecutive execution of the aggregation operation after discretization can be inefficient both in terms of space needed to log all elements of each window as windows can be very large in size. At the same time, the response time has to be minimized when iterating through all window contents in order to calculate an aggregate. Finally, and most importantly, as highlighted in Fig. 2, sliding windows might involve a large amount of overlapping across consecutive windows. In this example there is an overlapping of three records between every two windows.

The first problem is solved by simply pipelining discretization with aggregation and thus, effectively providing a partial evaluation of window aggregates. Partial aggregation is described in section "Partial Window Aggregation". The overlapping problem is more complex, and its optimization techniques are further classified by window types into slicing, general pre-aggregation, and hybrid techniques, covered thoroughly in sections "Window Slicing", "General Pre-aggregation", and "Cutty: A Hybrid Approach", respectively.

Partial Window Aggregation

A complete partial window aggregation scheme has been proposed in both FlatFat (Tangwongsan et al., 2015) and Cutty (Carbone et al., 2016). According to that scheme, an aggregation f_a can be decomposed into partial aggregation operations in order to pipeline the process during discretization. A window aggregation function is therefore decomposed into `lift` and `lower` and a `combine` functions as such:

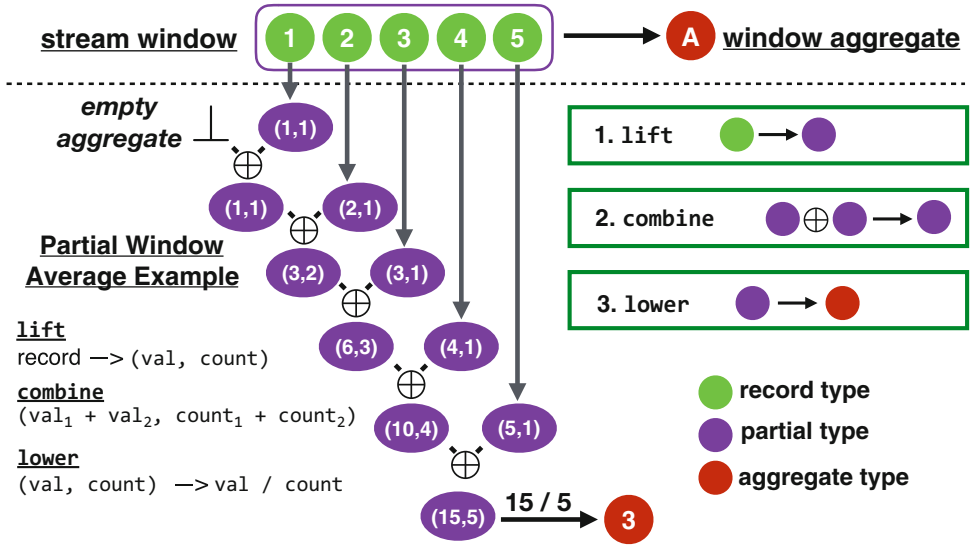
`lift` : $T \rightarrow A$ maps an element of a window to a partial aggregate of type A.

`combine` $\oplus : A \times A \rightarrow A$ combines two partial aggregates into a new partial aggregate (equivalent to a `reduce` function).

`lower` : $A \rightarrow T'$ maps a partial aggregate into an element in the type T' of output values.

The main and only requirement for partial aggregation is to have an *associative* `combine` function so that aggregation can be used to evaluate a full window aggregate in discrete steps (Arasu and Widom, 2004; Krishnamurthy et al., 2006; Tangwongsan et al., 2015).

An example of partial aggregation of a window is depicted in Fig. 3. The goal in this example is to partially compute the average value out of a set of records with values 1 to 5. The invariant is that only one partial aggregate is kept in memory (initially an empty aggregate). That aggregate is incrementally updated by each record that arrives in a window. To compute an average, two values have to be maintained in the aggregate type: a *sum* and a *count*. `lift` function, each record is first mapped into an aggregate type of its value



Stream Window Aggregation Semantics and Optimization, Fig. 3 Partial aggregation example for window average

and a count of 1. The `combine` function updates the partial aggregate with the new sum and count until all elements of a window have arrived. Then finally, the `lower` function transforms the aggregate into the window average, in this case this is 3.

Window Slicing

The amount of overlapping across sliding windows introduces additional space and computational complexity that partial aggregation itself cannot solve.

In the case of windows with predefined periodic characteristics such as a time or count slide, a family of optimization techniques is used to further decompose windows into nonoverlapping partial aggregates which can be shared and combined to calculate full window aggregates. This technique is typically named "slicing" or "bucketing." The two most popular slicing techniques that have also been deployed in production stream processing systems in the past are *panes* (Li et al., 2005a) and *pairs* (Krishnamurthy et al., 2006).

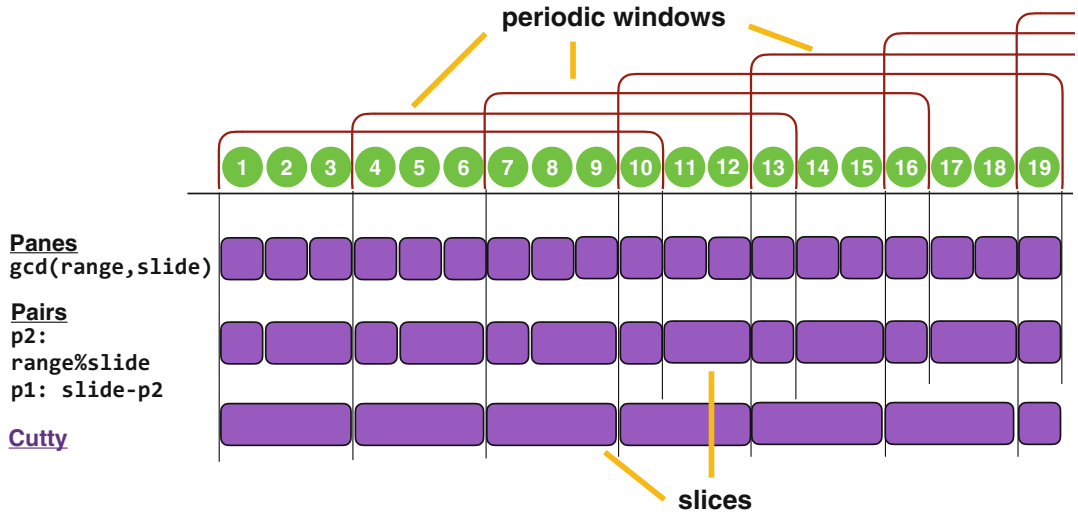
Panes

The main idea behind *Panes* is that if we have a periodic window query with a fixed slide and a range, it is trivial to break down the aggregation process into partials with a constant size, equal to the greatest common denominator of the respective *range* and *slide*. For example, if we have a sliding window with a range of 9 min and a slide of 6 min, the stream would be sliced and pre-aggregated into buckets, each of which corresponds to 3 min of the ingested stream (greatest common denominator of 6 and 9).

Panes have been criticized (Krishnamurthy et al., 2006) for their lack of general applicability, yielding unbalanced performance that depends highly on the combination of range and slide. For example, a window of range 10 s and a slide of 3 s would break down to slices of a single second, no longer exploiting the amount of nonoverlapping segments in a stream (as depicted in Fig. 4).

Pairs

The *pairs* technique (Krishnamurthy et al., 2006) splits a stream into two alternating slices: $p_2 = range \mod slide$ and $p_1 = slide - s_2$. This technique utilizes better nonoverlapping segments in a stream and seems to work well with most combinations of range and slide. Intuitively, pairs break slicing only when a stream window starts or ends. This is visualized in Fig. 4 where



Stream Window Aggregation Semantics and Optimization, Fig. 4 Example of different slicing techniques

it results into a lower number of slices for the same window compared to using *panes*. Contrary to *panes*, *pairs* has also been proposed for multi-query aggregation sharing where a large sequence of shared slices is decided at compilation time across shared sliding window aggregation queries in the same operator.

Slicing Limitations

While slicing offers the best known space and computational performance in sliding window aggregation, its applications are limited to periodic window queries since this is the only type of windows for which their beginning and end are predefined. The *Cutty* technique (Carbone et al., 2016) which is further analyzed in section "Cutty: A Hybrid Approach" avoids this strong assumption by letting user-defined functions signify during runtime when windows start or end. In the same work slicing is also combined with general pre-aggregation techniques (analyzed in section "General Pre-aggregation") in order to combine the strongest characteristics of both domains of window optimization.

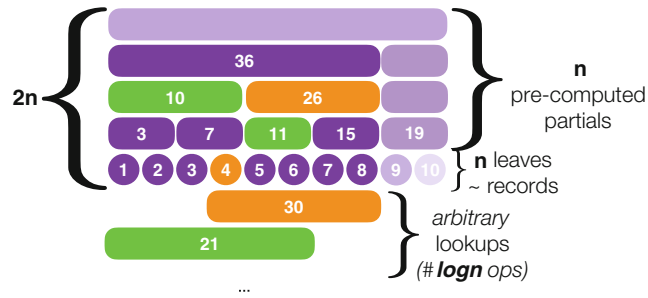
General Pre-aggregation

The main incentive of general pre-aggregation techniques is to be able to allow for arbitrary segment lookups in a stream (e.g., from external user queries) as depicted in Fig. 5.

The earliest work on general pre-aggregation was presented in *B-int* (Arasu and Widom, 2004) which precomputes eagerly higher-order partials on different segments of a stream. The application of B-int was meant to be fast aggregate retrieval for ad hoc stream queries (i.e., using CQL); however, the applicability of general pre-aggregation makes such techniques convenient for aggregating continuous sliding windows without a known range or slide or other periodicity assumptions. The Reactive Aggregator by IBM Research (Tangwongsan et al., 2015) exploits the properties of *B-Int* and introduces FlatFat: a fixed size circular heap binary tree of higher-order partials that "slides" together with the records of the stream.

General Pre-aggregation Limitations

General pre-aggregation offers fast retrievals of arbitrary windows on a stream at the cost of additional space and incremental update computation requirements. That is due to the fact that every time a new aggregate is added to the binary tree a sum of $\log(N)$ additional partial aggregations need to be employed in order to update all higher-order aggregates of the tree (given N: the number of active records/leaves in the tree). In summary,



Stream Window Aggregation Semantics and Optimization, Fig. 5 Example of a general pre-aggregation tree

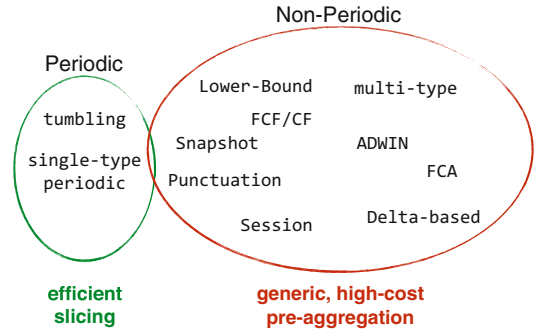
the runtime costs of employing eager aggregation, which are also visualized in Fig. 5 are the following:

space: $2N$ partials need to always be kept on heap to hold the full aggregation binary tree.

update/lookup: both update and full window lookup have $O(\log N)$ computational complexity. In the case of updates, the complexity is fixed ($\log N$) against slicing which typically involves a single aggregation per record.

All these costs pose an interesting trade-off when eager aggregation is employed per record in a data stream, which often results in more operations than a naive execution of each redundant window aggregation separately. As a result, it is likely that if general pre-aggregation is de facto applied, its runtime cost would never be amortized across a full run of a continuous application.

Nevertheless, the power of general pre-aggregation lies at the observation that it can be generally applied to any type of windows, thus, covering a large space of nonperiodic window types used within research and industrial applications, as depicted in Fig. 6.



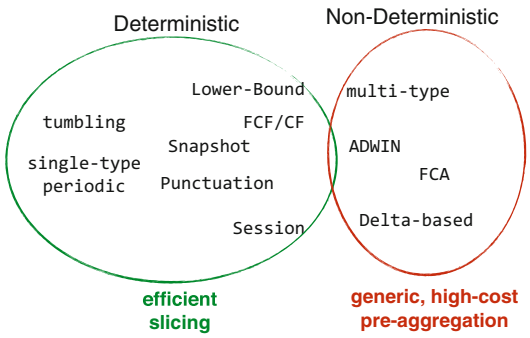
Stream Window Aggregation Semantics and Optimization, Fig. 6 Applicability of slicing and general pre-aggregation

Cutty: A Hybrid Approach

Slicing and general pre-aggregation are orthogonal techniques that can be potentially combined to support a wider variety of stream windows for aggregation. The *Cutty* aggregator (Carbone et al., 2016) employs such a hybrid approach that can lead to efficient aggregation of a broader number of window types than simply periodic. The main observation behind its design is that there is an implicit class of windows (a superclass of periodic ones), termed *deterministic* which can be obtained by using the right core primitives in the programming model. Deterministic windows can be used to enable efficient shared aggregation without limiting window expressivity. Figure 7 shows the expressive power of deterministic windows, being able to provide optimal pre-aggregation to more than the limited periodic windows.

Deterministic Windows

The concept of deterministic windows stems from the observation that all it takes to achieve optimal slicing is not the a priori knowledge of the periodicity of windows (if any) but the *runtime* knowledge of where a new window starts. While partial aggregation is employed, as described in section "Partial Window Aggregation", a single partial result can be kept in active memory until we receive a record that marks the beginning of a new window. Conceptually, a new window start means that



Stream Window Aggregation Semantics and Optimization, Fig. 7 Visualizing the expressive power of Deterministic Windows for Efficient Aggregation

we will later need a partial aggregate (or slice) starting at that point to evaluate the window that started there.

Cutty proposes user-defined windowing through the use of a *discretization function* f_{disk}, defined as follows:

$$f_{\text{disc}} : T \rightarrow \langle W_{\text{begin}} : \mathbb{N}, W_{\text{end}} : \mathbb{N} \rangle$$

where for each record $r \in T$: (i) W_{begin} is the number of windows beginning with r and (ii) W_{end} the number of windows ending with r.

Overview of Cutty

Optimal slicing with deterministic windows minimizes but does not eliminate redundancy. As an example, consider the slices produced by Cutty during the example execution of Fig. 4. A detailed observation of the slices used per window reveals a level of redundancy that cannot be handled by slicing. For example, slices $s[4, 6]$ and $s[7, 9]$ would have to be combined together twice: once for computing $f_a(s[1, 10])$ itself and once for computing $f_a(s[4, 13])$. Instead, if somehow the evaluation of $f_a(s[4, 9])$ was stored, it would not be necessary to repeat that aggregation.

Cutty utilizes general pre-aggregation (FlatFat) in order to further reduce the cost of full window aggregate evaluation and compute higher-order combinations of aggregates only once. This idea gives a new purpose to general pre-aggregation techniques and grants a low memory footprint since the space complexity of the aggregation tree is bounded by the number of active slices (which is equivalent to the minimum number of nonoverlapping segments in a stream). A full example run of *Cutty* is visualized in Fig. 8, showing both slicing of deterministic windows and general pre-aggregation and evaluation on stored partials. In the same example, notice that the partial $P(s[6, 7])$ is computed once and reused for both $f_a(s[1, 5])$ and $f_a(s[1, 6])$.

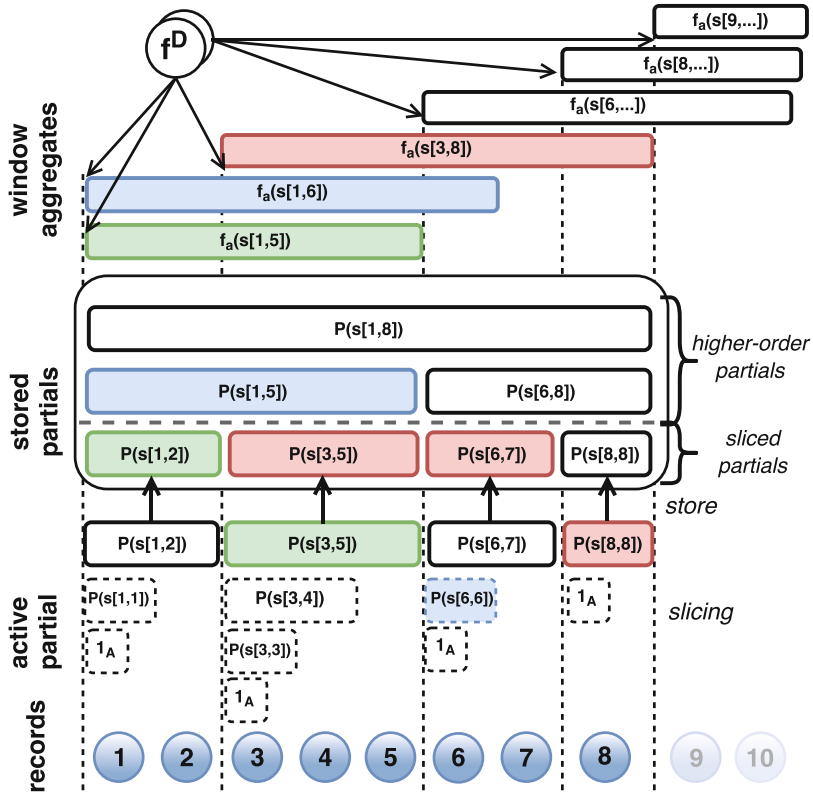
For nondeterministic windows, *Cutty* falls back to general pre-aggregation (simply using FlatFat) since slicing cannot be applied. Nevertheless, the generality and flexible (runtime specific) nature of this aggregation technique also enables the prospect of using it for applying operator sharing (Hirzel et al., 2014) on data streams. A full complexity and performance analysis and comparison are provided in the original *Cutty* paper (Carbone et al., 2016).

Further Works

Window aggregation is an interesting research topic, and there are many relevant proposed ideas to the ones presented here. For example, Li et al. (2005b, 2008a) classified window types by their evaluation context requirements, leaving the characterization of each class as an open research question. Performing certain types of aggregates in constant-time was recently proposed (Tangwongsan et al., 2017). Deterministic functions in *Cutty* subsume all forward-context-free windows (no future records are required to know when a window starts), while nondeterministic discretization functions are forward-context-aware. Heuristic-based plan optimizers have also been proposed (e.g., *TriWeave* Guirguis et al. 2012) to fine-tune the execution of periodic time queries dynamically using runtime metrics (i.e., input rate and shared aggregate rate).

Future Directions

Windowing semantics are becoming increasingly more complex and sophisticated as data stream



Stream Window Aggregation Semantics and Optimization, Fig. 8 An overview example of *Cutty*

processing is widely adopted. Aggregation techniques will have to follow the trends in windowing semantics and adapt to more dynamic, data-centric window types. One of the most prominent future directions in stream windowing is its standardization and encapsulation in stream SQL standards that are undergoing in open-source communities (e.g., the Calcite project and Google Dataflow Akidau et al. 2015). However, no significant efforts have been made to apply relational optimizations in stream windowing. Another future direction is to extend slicing capabilities beyond deterministic windows (if possible) and cover cases of fully data-driven windows without FIFO guarantees such as ADWIN (Bifet and Gavalda, 2007). Finally, general pre-aggregation data structures have to employ the notion of out-of-orderness (Traub et al., 2018). Currently, with existing out-of-the-box solution such as FlatFat, it is not possible to retract already evaluated window aggregates, thus, making it impossible to use for systems like Beam and Flink with out-of-order logic.

Conclusions

Windows over streaming data continue to be the most central abstraction in data stream processing. Aggregation techniques aim to reduce the computational redundancy to the maximum extent possible for sliding windows. Most often, approaches to efficient aggregation are entangled with actual windowing semantics, such as assuming periodic queries to provide efficient pre-aggregation. Slicing techniques provide low memory footprint and generally good performance at the cost of limited applicability, while general pre-aggregation techniques can be employed for any window lookup at the cost of high computational and

memory footprint. Recent approaches aim for a hybrid solution by generalizing slicing further while combining data structures from general pre-aggregation. Sliding window aggregation remains a challenging topic today, and new challenges will arise with the adoption of richer and more complex windowing semantics and out-of-order streams.

Cross-References

▸ Sliding-Window Aggregation Algorithms

References

Akidau T, Bradshaw R, Chambers C, Chernyak S, Fernández-Moctezuma RJ, Lax R, McVeety S, Mills D, Perry F, Schmidt E et al (2015) The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. In: VLDB

Arasu A, Widom J (2004) Resource sharing in continuous sliding-window aggregates. In: VLDB

Arasu A, Babcock B, Babu S, Cieslewicz J, Datar M, Ito K, Motwani R, Srivastava U, Widom J (2016) Stream:the Stanford data stream management system. In: Data stream management. Springer, Berlin/Heidelberg, pp 317–336

Arasu A, Babu S, Widom J (2006) The CQL continuous query language: semantic foundations and query execution. In: VLDBJ

Bifet A, Gavalda R (2007) Learning from time-changing data with adaptive windowing. In: SDM. SIAM

Botan I, Derakhshan R, Dindar N, Haas L, Miller RJ, Tatbul N (2010) Secret: a model for analysis of the execution semantics of stream processing systems. In: VLDB

Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K (2015) Apache Flink: stream and batch processing in a single engine. Bull IEEE Comput Soc Tech Commun Data Eng 36(4):28–38

Carbone P, Traub J, Katsifodimos A, Haridi S, Markl V (2016) Cutty: aggregate sharing for user-defined windows. In: Proceedings of the 25th ACM international on conference on information and knowledge management. ACM

Carbone P, Ewen S, Fóra G, Haridi S, Richter S, Tzoumas K (2017) State management in Apache Flink®: consistent stateful distributed stream processing. Proc VLDB Endow 10(12):1718–1729

Chandrasekaran S, Cooper O, Deshpande A, Franklin MJ, Hellerstein JM, Hong W, Krishnamurthy S, Madden SR, Reiss F, Shah MA (2003) TelegraphCQ: continuous dataflow processing. In: Proceedings of the 2003 ACM SIGMOD international conference on management of data. ACM, pp 668–668

Guirguis S, Sharaf MA, Chrysanthis PK, Labrinidis A (2012) Three-level processing of multiple aggregate continuous queries. In: IEEE ICDE

Hirzel M, Andrade H, Gedik B, Kumar V, Losa G, Nasgaard M, Soule R, Wu K (2009) SPL stream processing language specification. NewYork: IBMResearchDivisionTJ WatsonResearchCenter, IBM ResearchReport: RC24897 (W0911–044)

Hirzel M, Soulé R, Schneider S, Gedik B, Grimm R (2014) A catalog of stream processing optimizations. ACM Comput Surv (CSUR) 46(4):46

Krishnamurthy S, Wu C, Franklin M (2006) On-the-fly sharing for streamed aggregation. In: AMC SIGMOD

Li J, Maier D, Tufte K, Papadimos V, Tucker PA (2005a) No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. ACM SIGMOD Rec 34:39–44

Li J, Maier D, Tufte K, Papadimos V, Tucker PA (2005b) Semantics and evaluation techniques for window aggregates in data streams. In: ACM SIGMOD

Li J, Tufte K, Maier D, Papadimos V (2008a) Adaptwid: an adaptive, memory-efficient window aggregation implementation. IEEE Internet Comput 12:22–29

Li J, Tufte K, Shkapenyuk V, Papadimos V, Johnson T, Maier D (2008b) Out-of-order processing: a new architecture for high-performance stream systems. Proc VLDB Endow 1(1):274–288

Tangwongsan K, Hirzel M, Schneider S, Wu KL (2015) General incremental sliding-window aggregation. In: VLDB

Tangwongsan K, Hirzel M, Schneider S (2017) Low-latency sliding-window aggregation in worst-case constant time. In: Proceedings of the 11th ACM international conference on distributed and event-based systems. ACM, pp 66–77

Traub J, Grulich P, Rodriguez Cuellar A, Bress S, Katsifodimos A, Rable T, Markl V (2018) Scotty: efficient window aggregation for out-of-order stream processing. In: 2012 IEEE 34th international conference on data Engineering (ICDE). IEEE

S

Stream-Based Microservices

▸ Streaming Microservices

Streaming Architecture

▸ Streaming Microservices

Streaming Big Spatial Data

Ahmed R. Mahmood and Walid G. Aref
Purdue University, West Lafayette, IN, USA

Synonyms

Distributed spatial data streaming; Real-time big spatial data processing

Definitions

Recently, several big data stream management systems (DSMS, for short) have been developed to provide an infrastructure to process streamed big data. Big spatial DSMSs constitute a special class of big DSMSs that are optimized to process large amounts of spatial data streams. The main idea behind most big spatial DSMSs is to leverage the spatial properties of the incoming data stream to fairly distribute the workload across multiple distributed processes. When processing big spatial data streams, it is important to maintain high throughput and low latency.

Overview

Spatial data is ubiquitous. It is continuously being generated at a large scale. This is due to the popularity of GPS-enabled devices, e.g., smartphones, smart-watches, personal activity trackers, and GPS-navigation devices. Efficient processing of this streamed big spatial data requires higher computational resources than the resources that exist in a centralized system.

Big spatial DSMSs can be classified into the following two main categories: (1) *dedicated big spatial DSMSs* and (2) *spatial extensions to general-purpose big DSMSs*.

- *Dedicated big spatial DSMSs.* These systems are the first step toward scalable spatial data stream processing. These distributed big spatial DSMSs are built from scratch with the sole purpose of processing streamed big spatial data. These systems often lack important features that exist in general-purpose big DSMSs, e.g., fault tolerance, and the ability to perform non-spatial data processing. Also, designers of these systems implement the communication protocols between the distributed processes. These communication protocols often require serialization, synchronization, and encryption. These requirements are orthogonal to big spatial data stream processing and require a substantial development overhead.
- *Spatial extensions to general-purpose big DSMSs.* In this widely adopted approach, general-purpose big DSMSs, e.g., Storm (Toshniwal et al., 2014), Spark-Streaming (Zaharia et al., 2012), and Yahoo S4 (Neumeyer et al., 2010), are extended with spatial distribution approaches to optimize the processing of spatial queries over spatial data streams. When extending an existing big DSMS with spatial capabilities, one inherits the features of the underlying general-purpose big DSMS, e.g., high scalability, fault-tolerance, and robust communication protocols.

The streamed spatial data is often associated with textual data, e.g., as in the case of tweets. A tweet contains textual data as well as the user's location where the tweet is issued. Big spatio-textual DSMSs are designed to account for the spatial and the associated textual attributes to optimize the processing of spatio-textual queries over spatio-textual data streams.

Key Scientific Findings

First, we describe the main query types over spatial data streams. Then, we give a historical background on general-purpose big DSMSs and centralized spatial DSMSs. Then, we focus on the categories of big spatial DSMSs.

Query Types over Spatial Data Streams

There is a wide range of spatial predicates that can be used when querying spatial data streams. The two main classes of spatial predicates are spatial selects and spatial joins.

1. *Spatial Select*: In a spatial select, given a spatial data stream and a spatial predicate, the spatial select produces as output the data tuples from the spatial data stream that satisfy the spatial predicate. Examples of a spatial select are the spatial range select and the top-k nearest neighbor select.
   The range select predicate operates on a single spatial data stream and has a specific spatial range predicate. This select predicate identifies the streamed data tuples (i.e., the streamed spatial objects, e.g., cars or shops) that satisfy the spatial range predicate, i.e., the spatial objects that are located inside the specified spatial range.
   The k-nearest-neighbor (kNN) select predicate operates on a single spatial data stream and has a focal point and a parameter k. The kNN spatial select identifies the k-nearest data objects to the focal point specified. The distance function used in measuring the distance from the focal point to the spatial data objects can vary. It can be the Euclidean distance or the distance over an underlying road network.
2. *Spatial Join*: In a spatial join, given two spatial data streams and a spatial join predicate, the spatial join produces as output data tuple pairs, one from each spatial data stream, such that each pair of tuples satisfies the spatial join condition. The spatial join has many flavors based on the type of spatial join predicate that is being used. The following are two popular spatial join operation.
   The spatial range join: The spatial predicates can be of various forms. For example, one spatial predicate can be a distance threshold, say d. In this case, the spatial join will produce as output the pairs of objects, say (x_i, y_j), where Object x_i from the first data source is within distance d from Object y_j from the second data source. This type of operation is referred to as a spatial range join operation.
   The spatial k-nearest neighbor join (kNN-join): Depending on the spatial predicate involved, other types of spatial joins may be formed. For example, the spatial kNN-join operation produces as output the pairs of objects, say (x_i, y_j), where Object y_j from the second data source is among the k-closest objects to Object x_i from the first data source. In addition to the two input spatial data streams, the number k is also specified as an input parameter to the spatial kNN-join operation. Any distance function can be used. Examples are the Euclidean or Manhattan distances or the distance over an underlying road network graph.

A single spatial data source may be used in the spatial join in contrast to two data sources. In this case, it is termed a spatial self-join. For example, in a spatial range self-join, pairs of spatial objects from the same source that are within distance d from each other are reported as output. For all the above operations, various distance functions can be used, e.g., the Euclidean, Manhattan, and road network distance metrics.

Continuous vs. Snapshot Spatial Queries

Spatial queries over data streams can either be evaluated as snapshot or continuous queries. A snapshot query operates on the current state of the spatial stream and produces a single resultset. In contrast, a continuous spatial query operates continuously for a relatively long duration, where the resultset of the continuous query is continuously updated to reflect the incoming spatial data from the streamed spatial data source.

To evaluate a snapshot spatial query, the state of the streamed spatial objects is maintained (usually using a spatial index). For example, consider a stream of location updates of moving objects. The state of the stream could be the current locations of the moving objects or a limited history of the trajectories of the moving objects. When a snapshot query arrives, it is evaluated once over the current state of spatial data stream and produces as output the spatial data objects that qualify the query. Finally, the snapshot query is completely evaluated, and it quits.

In contrast, when a continuous query arrives, it is stored into the system and is progressively evaluated against the incoming spatial data objects. The DSMS maintains a state for each of the continuous queries. This state may include some intermediate summary data as well as the current resultset of the query. For example, in the case of continuous query that contains a kNN select predicate, the DSMS maintains the current set of k spatial objects that are closest to the focal point. Because the query is continuous, as the objects move, the current set of k-nearest objects to the focal point continues to get updated.

Continuous spatial queries are often indexed (using a spatial index) to speed up the maintenance of the arriving spatial objects, e.g., the continuous updates of the locations of the spatial objects as they move in space. Having a spatial index facilitates the progressive evaluation of the continuous spatial query.

Static vs. Dynamic Spatial Queries

Continuous spatial queries over spatial data streams can either be static or dynamic. A static continuous spatial query does not change its input over time. For example, the spatial location of a kNN select predicate remains constant over time for the entire lifetime of the continuous spatial query. In contrast, a dynamic continuous spatial query may change its input parameters over time. For example, consider the following query: Find the three-closest gas stations to my current location while I am driving on the highway. This continuous query is composed of a kNN select predicate, where the focal point of the kNN select corresponds to the location of a moving object. Changes to the location of this focal point as the object moves may result in updating the query's resultset even without having any updates from the spatial data source.

Historical Background

General-Purpose Big DSMSs

Several general-purpose big stream management systems have been extended to support big spatial data stream processing. These systems provide an infrastructure for the scalable processing of data streams. Generally, the main advantages of these systems are (1) hiding the complexity of transferring data between distributed processes, (2) high throughput, and (3) reliability and fault tolerance. Two main data processing models are currently being adopted by general-purpose big DSMSs, namely, *tuple-based* and *micro-batching*.

In the *tuple-based* streaming model, every tuple in the stream can be processed individually. Yahoo S4 (Neumeyer et al., 2010) and Apache Storm (Toshniwal et al., 2014) are examples of tuple-based big data streaming systems. The main advantage of the tuple-based model is the low processing latency of streamed tuples. The main limitation of this model is that a single tuple may be processed more than once in certain failure situations.

In the *micro-batching* streaming model, tuples are not processed individually but are rather buffered for a time duration (up to a few seconds) to create a micro-batch. M3 (Aly et al., 2012) is a main-memory map-reduce-based system that modifies (Apatche Hadoop, 2017) in the following way. It removes the HDFS layer and replaces it by memory buffers that are maintained in a distributed layer of data nodes. Data continues to accumulate until the memory buffers are filled or until some maximum delay threshold is reached. Then, data is batched into the map-reduce layer. In M3, the disk layer between the mappers and the reducers is eliminated and is replaced by another layer of distributed memory buffers. Data is communicated between the mappers' and the reducers' distributed memory buffers using networking sockets. SparkStreaming (Zaharia et al., 2012) is another example of a micro-batching-based DSMS. SparkStreaming extends Spark, a general-purpose main-memory big data system. SparkStreaming inherits the resilient distributed datasets (RDD) and lineage from Spark to ensure fault tolerance in stream processing. RDD is an immutable main-memory data structure that represents a collection of objects. Processing in Spark and SparkStreaming is performed by applying transformations to RDDs. Lineage is a graph that represents transformations to RDDs. In failure scenarios, the lineage graph is consulted

to reconstruct the failed RDDs. SparkStreaming builds RDDs over micro-batches of streams every specific time duration (up to a few seconds). The main advantages of this model are high throughput and that every tuple gets processed exactly once even under failure scenarios. However, the main limitation of micro-batching is the high latency incurred while buffering streamed data to build a micro-batch.

Centralized Spatial DSMSs

Several centralized spatial data stream management systems have been developed to answer spatial queries over spatial streams. Examples of these systems include PLACE (Mokbel et al., 2004b), SINA (Mokbel et al., 2004a), SEA-CNN (Xiong et al., 2005), SOLE (Mokbel and Aref, 2008), and Gpac (Mokbel and Aref, 2005). These systems are not designed for scalability, and their performance is constrained by the resources of the single machine they run on.

Big Spatial DSMSs

We describe the main types of big spatial DSMSs below.

Dedicated Big Spatial DSMSs

Special-purpose big spatial DSMSs are distributed systems that can process only spatial data streams. These systems often address specific types of queries over spatial data streams. They focus on the scalability of spatial query processing and do not address other requirements of distributed big data processing, e.g., the efficient scheduling of distributed processes across the cluster of machines, reliability, fault tolerance, error recovery, or fair workload distribution across the processes. These systems are considered the first step toward big spatial DSMSs. Examples of these dedicated and spatial-purpose distributed big spatial DSMSs include PLACE* (Xiong et al., 2007), MobiPLACE (Zakhary et al., 2013), and MobiEyes (Gedik and Liu, 2006).

*PLACE** (Xiong et al., 2007) use multiple servers to evaluate continuous moving range queries (dynamic queries) over moving objects in the Euclidean space. This system employs an incremental update model that updates query results only when there is a change to the resultset of the queries. MobiPLACE (Zakhary et al., 2013) applies the processing model of PLACE* over road networks.

MobiEyes (Gedik and Liu, 2006) use a centralized server and multiple small smart-mobile devices. To reduce the workload at the centralized server, MobiEyes offloads and distributes the processing of the continuous moving spatial range queries to the mobile devices.

Spatial Extensions to Big DSMSs

The most adopted approach in big spatial data streaming is to extend a general-purpose big data stream management system. This allows spatial extensions to inherit the scalability and reliability features of the underlying streaming platform. Spatial extensions to general-purpose big data stream management systems differ in the streaming model supported, i.e., tuple based or micro-batching based. These systems also differ in the type of spatial queries supported, the type of the spatial indexes used, and whether the spatial extension is adaptive and sensitive to the change in distribution of the data and the query workload of the spatial data stream or not.

Micro-Batching-Based Systems

Grid-based indexing on top of SparkStreaming (Choi et al., 2015; Lee and Song, 2015). These systems answer continuous range queries over spatial data streams on top of SparkStreaming. They use grid-based partitioning to distribute the streamed workload across the worker processes of the underlying cluster.

Cruncher (Abdelhamid et al., 2016). This system is an adaptive extension to SparkStreaming that addresses kNN and range queries over a spatial data stream. Cruncher keeps statistics about the spatial data and query workload to build a kd-tree-based (Ooi et al. 1987) spatial partitioning over the spatial data and queries. This partitioning is applied to every micro-batch to create

sub-micro-batches. These sub-micro-batches are fairly distributed across the worker processes of the underlying spark cluster. Cruncher also uses the spatial properties of continuous range queries to share the execution among the spatial queries that have overlapping ranges.

Tuple-Based Systems

Algorithms for spatial queries on top of Storm (Zhang et al., 2016). This approach develops algorithms to answer continuous spatial queries on top of Storm. The queries supported are the spatial range, kNN, and the spatial join. The current locations of the moving objects are maintained in separate Storm processes. Each Storm process is responsible for a specific set of moving objects. Within every Storm process, say P, a spatial index is used to maintain the current locations of the moving objects handled by P. Spatial queries are replicated to all Storm processes to produce partial results. Partial results of these queries are collected from the distributed Storm processes to be aggregated at dedicated aggregation processes. To process a spatial join operation, a different partitioning algorithm is used. Instead of partitioning the moving objects based on their identifiers, moving objects are partitioned spatially using grid-based partitioning. The range of a spatial join operation may span multiple Storm processes. Moving objects that overlap the spatial range of the spatial join operation are paired together to produce the final answer.

Continuous kNN-join processing on top of Storm (Song, 2016) This approach develops a parallel algorithm to evaluate continuous kNN-join operations on top of spatial streams. The algorithm uses data partitioning and replication of distributed processes to group in the same processes all the possible pairs that can be joined together. This algorithm is realized on top of Storm.

DSI: kNN processing on top of S4 (Yu et al., 2015). DSI is a distributed spatial partitioning mechanism that is realized on top of Yahoo S4 (Neumeyer et al., 2010). DSI processes snapshot kNN operations over the current locations of moving objects. DSI partitions the space into horizontal and vertical strips. Every horizontal and vertical strip is assigned to a distributed process in S4. To process a kNN operation, partial kNN results are calculated from the strips. Then, a global kNN result is aggregated from all the partial results. DSI is adaptive and attempts to ensure fair workload distribution by updating the boundaries of the horizontal and vertical strips based on the changes in the workload.

kNN processing on top of IBM System S (Wu et al., 2012). In this approach, the continuous kNN operation is handled as a stateful operator, i.e., it maintains the current set of the k-nearest objects closest to the focal point of the continuous kNN operation. A distributed version of the kNN operation is realized using IBM System S. IBM System S is a proprietary general-purpose big data stream management system. To evaluate this kNN operation, location updates of moving objects are partitioned using a hash function to multiple processes. Then, an aggregation step is used to find the global kNN resultset.

Big Spatio-textual Data Streaming

Spatial data streams are often associated with a textual attribute. For example, tweets have both spatial and textual attributes. Several big spatial data stream management systems have been developed to process streamed spatio-textual data. The main feature in these systems is that they utilize the textual attribute of the spatio-textual data to optimize the processing of spatio-textual queries.

The two main spatio-textual queries being supported by big spatial data stream management systems are (1) *the continuous spatio-textual filter query* and (2) *the continuous top-k spatio-textual query*.

The continuous spatio-textual filter query has both an input spatial range and a set of input keywords. In this query, it is required to identify the streamed spatio-textual objects located inside the spatial range of the query and contain all of the keywords of the query. Tornado (Mahmood et al., 2015, 2017) and PS^{2}Stream (Chen et al., 2017) are two big spatial data stream management sys-

tems that address these queries. Both systems extend Storm. The main idea behind these systems is to evenly distribute the spatio-textual data and queries over distributed Storm processes. This is performed using a global distribution layer that contains a spatio-textual distribution index. Within every worker process, continuous spatio-textual filter queries are indexed using an internal spatio-textual index.

The continuous top-k spatio-textual query. This query has an input focal point and an associated set of input keywords. This query maintains a time-sliding window over a spatio-textual data stream. In this query, it is required to continuously identify the most relevant spatio-textual data objects within the time-sliding window. Relevance between the objects and the query is calculated using a function of the spatial distance and the textual similarity between the spatio-textual data objects and the queries. DSkype (Wang et al., 2017) is an extension to Storm. DSkype uses a global spatio-textual distribution mechanism to distribute stream spatio-textual queries to worker processes. In worker processes, a local index stores the spatio-textual objects for a specific time-sliding window. For every incoming query, the distributed local indexes are searched to identify an initial list of the top-k relevant data objects. Then, the list of the top-k relevant data objects keeps being updated according to the incoming spatio-textual streamed data objects.

Examples of Applications

Many applications require the processing of this streamed big spatial data in real time. Example applications include finding travel companions; ride-sharing, e.g., Uber; real-time navigation and map services, e.g., Google Maps; and real-time traffic analysis. Nowadays, algorithms adopted in online advertisement targeting consider the locations of users to identify relevant local advertisements. Real-time analysis of micro-blogs, e.g., identify trending Twitter hashtags within a specific location, is one important application of processing streamed spatio-textual data.

Future Directions for Research

One important research direction is online big spatial data analytics. The challenge here is to support a richer set of complex spatial predicates that also involves aggregations. One challenge of supporting these spatial predicates is to devise effective cost estimation techniques that can capture the varying overhead of these operations. These cost estimation techniques are crucial to maintaining workload balance over the distributed processes of the underlying spatial DSMS.

Also, existing big spatial DSMSs do not efficiently support the temporal dimension. One example scenario of spatio-temporal and textual processing is the continuous aggregation of streamed spatial data based on their textual attributes over a time-sliding window. This type of data management requires novel distributed spatial-temporal and textual distribution, indexing, and query processing algorithms that account for the peculiarities of the temporal dimension.

Cross-References

- Apache Samza
- Apache Spark
- Spatio-textual data

S

References

Abdelhamid AS, Tang M, Aly AM, Mahmood AR, Qadah T, Aref WG, Basalamah S (2016) Cruncher: distributed in-memory processing for location-based services. In: IEEE 32nd international conference on data engineering (ICDE). IEEE, pp 1406–1409

Apatche Hadoop (2017) Apatche Hadoop. http://hadoop.apache.org/

Aly AM, Sallam A, Gnanasekaran BM, Nguyen-Dinh LV, Aref WG, Ouzzani M, Ghafoor A (2012) M3: stream processing on main-memory mapreduce. In: ICDE, pp 1253–1256

Chen Z, Cong G, Zhang Z, Fuz TZ, Chen L (2017) Distributed publish/subscribe query processing on the

spatio-textual data stream. In: IEEE 33rd international conference on data engineering (ICDE). IEEE, pp 1095–1106
Choi D, Song S, Kim B, Bae I (2015) Processing moving objects and traffic events based on spark streaming. In: 8th international conference on disaster recovery and business continuity (DRBC). IEEE, pp 4–7
Gedik B, Liu L (2006) Mobieyes: a distributed location monitoring service using moving location queries. IEEE Trans Mobile Comput 5(10):1384–1402
Lee Y, Song S (2015) Distributed indexing methods for moving objects based on spark stream. Int J Contents 11(1):69–72
Mahmood AR, Aly AM, Qadah T, Rezig EK, Daghistani A, Madkour A, Abdelhamid AS, Hassan MS, Aref WG, Basalamah S (2015) Tornado: a distributed spatio-textual stream processing system. PVLDB 8(12): 2020–2023
Mahmood AR, Daghistani A, Aly AM, Aref WG, Tang M, Basalamah S, Prabhakar S (2017) Adaptive processing of spatial-keyword data over a distributed streaming cluster. arXiv preprint, arXiv:170902533
Mokbel MF, Aref WG (2005) Gpac: generic and progressive processing of mobile queries over mobile data. In: Proceedings of the 6th international conference on mobile data management. ACM, pp 155–163
Mokbel MF, Aref WG (2008) Sole: scalable on-line execution of continuous queries on spatio-temporal data streams. VLDB J 17(5):971–995
Mokbel MF, Xiong X, Aref WG (2004a) Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In: Proceedings of the 2004 ACM SIGMOD international conference on management of data. ACM, pp 623–634
Mokbel MF, Xiong X, Aref WG, Hambrusch SE, Prabhakar S, Hammad MA (2004b) Place: a query processor for handling real-time spatio-temporal data streams. In: Proceedings of the thirtieth international conference on very large data bases, VLDB endowment, vol 30, pp 1377–1380
Neumeyer L, Robbins B, Nair A, Kesari A (2010) S4: distributed stream computing platform. In: IEEE international conference on data mining workshops (ICDMW). IEEE, pp 170–177
Ooi BC, McDonell KJ, Sacks-Davis R (1987) Spatial kd-tree: an indexing mechanism for spatial databases. In: IEEE COMPSAC, sn. vol 87, p 85
Song G (2016) Parallel and continuous join processing for data stream. PhD thesis, Université Paris-Saclay
Toshniwal A, Taneja S, Shukla A, Ramasamy K, Patel JM, Kulkarni S, Jackson J, Gade K, Fu M, Donham J et al (2014) Storm@ twitter. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. ACM, pp 147–156
Wang X, Zhang W, Zhang Y, Lin X, Huang Z (2017) Top-k spatial-keyword publish/subscribe over sliding window. VLDB J 26(3):301–326
Wu S, Kumar V, Wu KL, Ooi BC (2012) Parallelizing stateful operators in a distributed stream processing system: how, should you and how much? In: Proceedings of the 6th ACM international conference on distributed event-based systems. ACM, pp 278–289
Xiong X, Mokbel MF, Aref WG (2005) SEA-CNN: scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In: Proceedings of the 21st international conference on data engineering, ICDE 2005. IEEE, pp 643–654
Xiong X, Elmongui HG, Chai X, Aref WG (2007) Place: a distributed spatio-temporal data stream management system for moving objects. In: International conference on mobile data management. IEEE, pp 44–51
Yu Z, Liu Y, Yu X, Pu KQ (2015) Scalable distributed processing of k nearest neighbor queries over moving objects. IEEE Trans Knowl Data Eng 27(5):1383–1396
Zaharia M, Das T, Li H, Shenker S, Stoica I (2012) Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. HotCloud 12:10–10
Zakhary V, Elmongui HG, Nagi MH (2013) Mobiplace*: a distributed framework for spatio-temporal data streams processing utilizing mobile clients processing power. In: International conference on mobile and ubiquitous systems: computing, networking, and services. Springer, pp 78–88
Zhang F, Zheng Y, Xu D, Du Z, Wang Y, Liu R, Ye X (2016) Real-time spatial queries for moving objects using storm topology. ISPRS Int J Geo-Inf 5(10):178

Streaming Language

▸ Stream Processing Languages and Abstractions

Streaming Microservices

Ted Dunning and Ellen Friedman
MapR Technologies and Apache Software Foundation, Santa Clara, CA, USA

Synonyms

Stream-based microservices; Streaming architecture

Definitions

Streaming microservices refers to a style of building large-scale software systems as a collection of independently deployable processes that commu-

nicate via persistent message streams rather than via remote procedure calls (RPC). A key distinction between streaming and RPC-based microservices is that streaming microservices provide a high degree of temporal decoupling between services. Not only does it not matter how a service does what it does, but it also matters much less exactly when it does it.

The use of streaming microservices is distinct from previous architectural trends such as Service Oriented Architecture (SOA) because the message streams used with streaming microservices are very simple, supporting little more than ordered delivery of messages organized into topics as opposed to the highly complex semantics supported by Enterprise Service Busses (ESB).

Historical Background

The term microservices as a style of systems architecture was introduced in 2011 at a conference in Venice and gained popularity around 2013–2014 (Fowler and Lewis, 2014) as people saw the success of this approach in large companies including Netflix, Amazon, and LinkedIn. Many of the core concepts underlying microservices for architecting large-scale systems go back at least a decade earlier, but the distinctive characteristics of microservice architectures were not well articulated until more recently.

The basic idea of microservices is to provide useful decompositions of large systems into independent services that could be started, stopped, and redeployed relatively independently. The key characteristic is that the implementation details of services are hidden. Virtues attributed to microservices include agility and flexibility in the development process by avoiding a monolithic approach with many development dependencies. Since services can be redeployed independently, the system is easier to upgrade progressively through evolutionary refinement of services one at a time. Teams working on different services can work to differing deadlines, thus avoiding many dependencies and schedule blocks.

In addition to the use of micro-services as a way to structure systems, they also define team organization. The result is that large systems devolve into a suite of loosely coupled small services, each built and run by a small, focused, cross-functional team. As originally viewed, microservices communicate via synchronous remote procedure calls (RPC), often implemented as REpresentation State Transfer (REST) (Fielding and Taylor, 2002) requests implemented over HTTP connections. More recently, asynchronous remote procedure calls have become much more common, especially in conjunction with node.js-based systems (Hughes-Croucher and Wilson, 2012).

The ability to deploy microservices independently makes development much easier than in a monolithic system or in a system that forces tight coupling between requestor and service. This is because individual services that can be upgraded independently also allow decoupling of development schedules. Additionally, using simple remote calls defined using systems like ProtoBufs, strict versioning of communication protocols is not necessary. Instead a more evolutionary soft-versioning approach can be taken that views all protocol changes as extensions and avoids complete redefinitions. Soft versioning of communication protocols provides additional decoupling of service development.

Decoupling of services also makes scaling easier, so explosive growth is a strong incentive to adopt microservice architectures. Selectively scaling individual services to match the scale of work that needs to be done by that service is much more efficient than scaling a monolithic service (Newman, 2015).

The final step to streaming microservices as opposed to RPC-based microservices is a much more recent development than the core concept of microservices. The fundamental addition that streaming brings is the idea that many services can actually be decoupled temporally as well as in terms of deployment. This sounds like a small change, but it substantially increases the scope of the microservice concept.

Foundations

Getting streaming microservices to work well requires a lexicon of high-level architectural patterns, a compatible underlying message transport, and methods for implementing the microservices themselves. This section describes each of these requirements in turn.

Emergence of Streaming Microservices

Widespread adoption of streaming microservices is a relatively recent trend in spite of the fact that the idea of moving data from one processing element to another is an old one (Johnston et al., 2004). What distinguishes modern streaming microservices from older dataflow concepts is that whereas dataflow was largely intended to build a system to execute a single computer program, a streaming microservices system is intended to support multiple high-level services that can be modified at any time. The idea of mutability of individual services leads directly to the requirement for hiding implementation details. Hiding implementation details such as timing leads to the use of persistent streams.

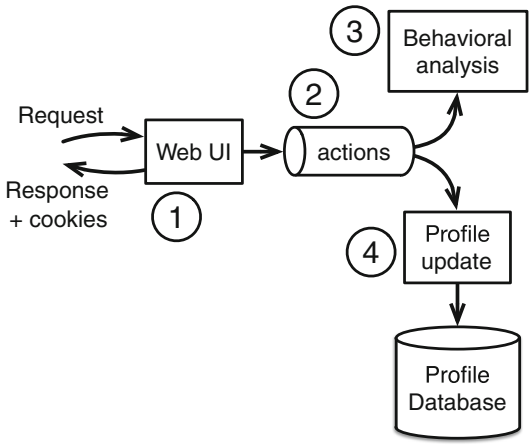
Existing messaging systems did not, however, support streaming microservices very well until recently. This prevented more than limited adoption of streaming microservices as a common architecture. This mirrors the way that early RPC mechanisms such as Java RMI and Corba were unsuitable for RPC-based microservices due to the way that changes in client or server were reflected through the remote call mechanism. True microservices could not really be built until more modern conventions for RPCs such as REST became widely accepted.

The problem with streaming microservices was that, until recently, message queuing systems put a premium on features that ensure that each message is processed exactly once. This exactly-once execution mirrors the way that a dataflow program executes. This mode of message consumption requires some kind of transaction per message and can severely limit the message rate and scalability (Videla and Williams, 2012; Snyder et al., 2011). More importantly, however, the transaction induces coupling between message producers and consumers and between different consumers, which degrades the isolation necessary for real microservices.

The release of Apache Kafka (Narkhede et al., 2017) provided an alternative approach to messaging that has proven more fertile for microservices. With Kafka, neither producers nor consumers have any idea how many consumers there are, and messages are kept in order and are retained for a configurable (and typically relatively long) period of time with no regard to whether they have been processed. Consumers are responsible for maintaining a history of which messages they have processed, although the framework does have a provision for consumers to record their consumption point back into the stream. Kafka guarantees that all consumers will see messages in the same order within the same partition of a topic but does little else to support the context of consumers. This same model, and indeed, the same API, have been adopted by commercial vendors such as MapR in their streams technology.

Pattern: Do Some, Defer Some

In microservice-based systems, some work typically has to be done by a service to produce a response, but it is also common that a significant amount of work involved in the request can be deferred to another time, possibly when a larger batch of such deferred tasks can be executed at the same time. It is a rather natural step to put the data associated with these deferred tasks into a message stream. Each message specifies a unit of work to be done when the message is read from the stream. Such a do-some/defer-some pattern is illustrated in Fig. 1. Here, we have three microservices: one is the web UI (labeled 1 in the figure) implemented as a synchronous RPC-based microservice, and the other two (labeled 3 and 4 in the figure) are implemented as streaming microservices. The web UI can respond to requests as quickly as possible and can have a simpler implementation because it only needs to focus on the minimal amount of processing necessary to provide a response. By recording a summary of the request it just processed as an event in a message stream, the web UI enables



Streaming Microservices, Fig. 1 In a typical design of a web service, the web UI (1) is implemented as a conventional microservice that accepts requests and returns responses in a synchronous fashion. In addition, the web UI defers some work by putting an event into a message stream (2). At a later time, other microservices such as a behavioral analysis (3) or a profile database update (4) process these events. Using streaming microservices here avoids impact on the latency of the original web UI request and improves the isolation of all three services

other microservices to do additional processing outside the critical path of responding to web UI requests. Because the web UI doesn't even necessarily know of the existence of these other services, the implementation and operation of the web UI are highly decoupled from them. This decoupling means that each of the three services can be updated and redeployed independently of the other services.

In this example, the web UI doesn't need to wait for confirmation that the work of the other services has completed nor does it rely on the actual results of that work. All it needs is certainty that the deferred work has been persisted in the message stream and will get done eventually. In fact, the web UI can often be implemented with no knowledge of these other microservices. The decoupling benefits of not depending on direct results and relaxed temporal dependency suggest that connecting services with message streams is a useful alternative to the more widely known RPC-based connected microservices. This style of design was called "event sourcing" by Fowler in Fowler (2005) and later called "streaming architecture" or "streaming microservices" (Dunning and Friedman, 2016).

One of the major advantages of streaming microservices over conventional RPC-based microservices is that streaming microservices allow temporal decoupling of services in addition to the ability to independently deploy and upgrade services. Temporal decoupling means that there is little dependency in time of computation for producers of messages and consumers. This decoupling allows major implementation details such as whether a producer or consumer runs intermittently or continuously to be hidden from other services. In the ultimate case of hiding implementation details, a consumer may not even have been implemented when messages are emitted by a producer. In that case, the producer cannot depend on the implementation details of the consumer since the consumer literally does not even exist when the message is produced by the producer.

Message Transport

The messaging technology used to communicate between streaming microservices has some key requirements that derive from the basic definition of microservices. Earlier messaging tools generally required a trade-off between performance and message persistence and induced coupling between producers and consumers, between producers on the same stream and between multiple consumers due to the semantics of the messaging system itself. These limitations made streaming microservices difficult to apply across a wide range of applications and thus substantially limited the uptake of streaming microservice architectures.

The effectiveness of the Kafka style of message streaming for microservices can seem a bit paradoxical; by providing fewer features and simpler semantics, it makes streaming microservices more practical. There are several major requirements that Kafka-esque systems satisfy, including:

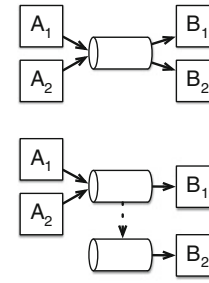
Persistence Messages are persisted in the message stream. This is logically required to enable service independence since there is no guarantee

that both producer and consumer are even running at the same time. Message persistence also improves failure recovery since a consumer can checkpoint input offsets and internal state, allowing clean restart on failure. Traditionally, however, persistence has been difficult to achieve at high performance levels.

Performance It is important that the message transport be fast enough for essentially any operational requirement. If this is not true and widely recognized, implementers will tend to proactively code around perceived performance risks. Wide adoption of streaming microservices can quickly lead to message throughputs in the millions of messages per second even at modest scale. It is not true, however, that all streaming microservices will require such high message rates. It is simply true that *some* applications will require such rates and using different streaming mechanisms for different applications or changing streaming technologies as services scale breaks down service independence because of a lack of pervasiveness.

Pervasive A key benefit of a microservices approach is that services can universally interoperate. For RPC oriented microservices, organizations typically standardize on a single RPC technology for this reason. For streaming microservices, similar standardization is required to gain adoption and allow interoperation of services. As such, any messaging system used has to meet technical requirements such as persistence and performance, but also social requirements like universal adoption within a set of services, that is, it must be pervasive. Put another way, if different services use different streaming technologies due to different performance levels, this introduces implementation coupling between systems that could be avoided with a single streaming system.

Ultimately, these properties of the underlying streaming technology result in a system that can allow services to function without knowledge of the implementation details of other services. In the upper part of Fig. 2, for example, services A_1 and A_2 can be producing data for B_1. Service B_2 can then be brought into operation without



Streaming Microservices, Fig. 2 Examples of decoupling between processes. In the top example, producers A_1 and A_2 and consumers B_1 and B_2 operate independently and can be deployed without affecting any of the other services. In the bottom example, A_1 and A_2 produce data that is consumed by B_1 and B_2 from different replicas of the same stream. Producers and consumers in the two examples can't tell the difference between the two configurations

B_1 or either producer being aware. In the lower example, geographical separation can also be introduced by replicating the stream without any of the services being affected, except possibly by small differences in the latency for messages arriving at B_1 and B_2.

Service Implementation

Streaming microservices can be implemented in a wide variety of ways. Particularly at the lower performance levels, it is possible to build reliable services using simple single-threaded code that reads units of work from a stream, performs the work and occasionally commits the current input point. This simple approach needs to be augmented with some sort of orchestrator such as Kubernetes that can restart or migrate the service as necessary. At slightly higher throughput levels, input streams can usually be partitioned to allow multiple processes to cooperate in a "consumer group." By subscribing as part of the same consumer group, these processes will be assigned different partitions. The worker processes have to handle out-of-band messages notifying them of changes of partition control, but the basic structure is extremely simple, at least as long as the necessary processing on each unit of work is relatively simple.

As the required complexity of a service's task goes up, it can be advantageous to use more

advanced systems to implement services. Options include Apache Spark's continuous streaming component (Apache Software Foundation, 2016b) or Apache Flink (Apache Software Foundation, 2016a; Friedman and Tzoumas, 2016). These more complex systems can handle complex joins and windowed transformations of streaming data while maintaining failure tolerance and high throughput.

Complementary Roles of the Message Stream and Database

In streaming microservices, message streams are used for communication between services, while databases are typically considered better for retaining state within a service. Often, a service will have a database that contains the latest state received via message stream. The sequence of states of such a database is, in some sense, equivalent to the messages in the stream. The primary difference is that the stream inherently retains the notion of time and all previous states but is difficult to probe for the latest value of any quantity. In contrast, the database makes probing for the latest value easy but loses information about previous states. As such, databases of this kind and the streams that update them can be seen as complementary views of the same data.

The virtue of storing state in a private database is that services can become decoupled in terms of time. If two consumers of a message stream update private databases in identical ways, there is no guarantee that at any given time that the two databases will have identical values because there is no guarantee that the two services will process messages at the same rate. We can guarantee, however, that both services will agree about the state of their databases relative to identical offsets in the message stream. This idea of equivalent, but not time-aligned, state is important because it allows much of the power of database transactions to be had without the abstraction destroying qualities of shared databases.

It should be noted that, as recommended by Fowler in his description of event sourcing, the messages in a message stream should generally be couched as heavily denormalized business-level events rather than table level updates. The concept of "business-level event" is not particularly well defined, but it is very close to the concept of REST in that the message should contain everything that is needed to describe the event in a self-contained way. The purpose of using RESTful messages of this type is that it is relatively easy to translate such messages into corresponding low-level table updates, but it is much more difficult to translate a collection of table updates into business-level event descriptions. This means that using RESTful messages makes it so that different services can easily have different data models in their private databases, which makes their implementations more loosely coupled.

Key Applications

The streaming style of microservices has widespread application, especially associated with more back-end systems such as analytical systems in which throughput and aggregated analytical results at scale are particularly important (Dunning and Friedman, 2016). Streaming microservice applications often involve situations in which delayed processing is acceptable.

In general, situations in which the flexibility of a stream-based overall architecture is desired may benefit from a streaming microservices style of work. An example where a streaming microservices style approach would be particularly useful is in analysis of IoT (Internet of Things) sensor data.

Cross-References

▸ Definition of Data Streams
▸ Rendezvous Architectures

References

Apache Software Foundation (2016a) Flink. http://flink.apache.org

Apache Software Foundation (2016b) Spark structured streaming. https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
Dunning T, Friedman E (2016) Streaming architecture: new designs using Apache Kafka and MapR streams. O'Reilly Media, Inc., Sebastopol
Fielding RT, Taylor RN (2002) Principled design of the modern web architecture. ACM Trans Internet Technol 2(2):115–150. http://doi.acm.org/10.1145/514183.514185
Fowler M (2005) https://martinfowler.com/eaaDev/EventSourcing.html
Fowler M, Lewis J (2014) Microservices. http://martinfowler.com/articles/microservices.html
Friedman E, Tzoumas K (2016) Introduction to Apache flink: stream processing for real time and beyond. O'Reilly Media, Inc., Sebastopol
Hughes-Croucher T, Wilson M (2012) Node – up and running: scalable server-side code with javascript. O'Reilly. http://shop.oreilly.com/product/0636920015956.do
Johnston WM, Hanna JRP, Millar RJ (2004) Advances in dataflow programming languages. ACM Comput Surv 36(1):1–34. https://doi.org/10.1145/1013208.1013209. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.99.7265
Narkhede N, Shapira G, Palino T (2017) Kafka – the definitive guide: real-time data and stream processing at scale. O'Reilly Media, Incorporated.http://shop.oreilly.com/product/0636920044123.do
Newman S (2015) Building microservices. O'Reilly Media, Inc., http://shop.oreilly.com/product/0636920033158.do
Snyder B, Bosanac D, Davies R (2011) ActiveMQ in action. Manning Publications Co., Greenwich. https://www.manning.com/books/activemq-in-action
Videla A, Williams J (2012) RabbitMQ in action: distributed messaging for everyone. Manning Pubs Co series. Manning Publications Company. https://www.manning.com/books/rabbitmq-in-action

Streaming Process Discovery and Conformance Checking

Andrea Burattin
DTU Compute, Software Engineering, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark

Synonyms

Online conformance checking; Online process discovery; Online process mining

Definitions

Streaming process discovery, streaming conformance checking, and streaming process mining in general (also known as *online process mining*) are disciplines which analyze event streams to extract a process model or to assess their conformance with respect to a given reference model. The main characteristic of this family of techniques is to analyze events immediately as they are generated (instead of storing them in a log for late processing). This allows to drastically reduce the latency among phases of the BPM lifecycle (cf. Dumas et al. 2013), thus allowing faster process adaptations and better executions.

Overview

A possible characterization of process mining algorithms is based on how they consume event data. Specifically, most of the algorithms focus on a (static) *event log*; however, there are algorithms which focus on *event streams*. An event log is a finite sampling of activities observed in a given time frame. An event stream, on the other hand, is an unbounded *sequence* of events, which contains events as they are executed.

Event streams, in fact, are specific types of data streams, where each data point refers to an event. General data streams have been investigated since many years, mainly to tackle problems such as frequency counting, classification, clustering, approximation, time series analysis, and change diagnosis (also known as novelty detection or concept drift detection) as summarized in Widmer and Kubat (1996), Gama (2010), Gaber et al. (2005), and Aggarwal (2007). To tackle these problems, it is possible to devise techniques in a data- or task-based orientation. *Data-based techniques* aim at extracting a significantly representative finite subset of the data stream, which is used for the analysis. *Task-based techniques*, on the other hand, adapt the computation to this new data modality, in order to minimize time and space complexity of the analysis.

The streams these algorithms have to deal with can be characterized based on their *operations model*. In particular, we might have:

- insert-only stream model (once an element is seen, it cannot be changed);
- insert-delete stream model (cf.n elements can be deleted or updated);
- additive stream model where seen item refers to numerical variables which are just incremented.

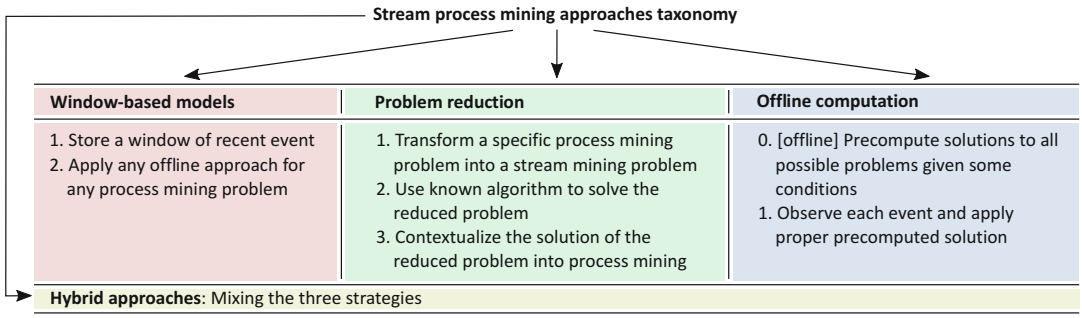
In the context of process mining, all available techniques assume the insert-only model: observations refer to activities which have been executed. The data mining literature, for example, in Golab and Özsu (2003) and Bifet et al. (2010), informally, defines data streams as a *fast sequence of data items*. However, in Bifet and Kirkby (2009), several assumptions on the data stream are reported, such as the following: the data is assumed to have a small and fixed number of attributes; the number of data points is very large; and the stream concepts (in the process mining context, the *concept* is the model underlying the events being generated) are assumed to be stationary or evolving. These assumptions make the processing of data streams very different from conventional relational models. Specifically, as detailed in Gama (2010, Table 2.1), the data elements arrive online, with no control by the system, in an unbounded amount. This imposes the analysis to be incremental: once an element is received, it is discarded or analyzed. If it is analyzed, it cannot be explicitly retrieved again afterward (i.e., its information is aggregated and summarized). Additionally, to cope with concept drifts, old observations should be replaced by new ones, and it is not possible to have one-time queries but a continuous "querying mechanism" is necessary.

Key Research Findings

This section provides some general ideas on how to tackle the problem of process mining from an event stream. Three general strategies are sketched, and in the upcoming subsections, details regarding actual instantiations of the ideas are reported.

Figure 1 depicts a taxonomy of the different approaches investigated so far. In this text, we will not focus on hybrid approaches: they are characterized by very heterogeneous solutions and therefore there is no proper generalization possible.

Window models. In order to perform process mining on a stream of event, the simplest approach is to devised a data-based technique to store only the set of most recent events observed and periodically analyze them. Whenever there is no more memory available, the oldest event is discarded. This approach, called *window model*, is described in Algorithm 1. The algorithm enters an endless loop where, at each iteration, a new event is observed. Then, the system checks whether it is necessary to remove old events or not, and



Streaming Process Discovery and Conformance Checking, Fig. 1 Taxonomy of the different approaches to solve the different stream process mining problems. For each technique, corresponding general steps are sketched

Algorithm 1: Window model

Input: S: event stream
M: memory model
max_M: maximum memory
A: additional information (e.g., a reference model), can be $\emptyset$

forever do
  // Observe a new event
  $e \leftarrow observe(S)$
  // Memory update
  if $max(M) \geq max_M$ **then**
    $dequeue(M)$ // Forgetting
  end
  $insert(M, e)$
  // Mining update
  if *perform mining* **then**
    // Memory into event log
    $L \leftarrow convert(M)$
    $ProcessMining(L, A)$
  end
end

Algorithm 2: Lossy counting

Input: S: data stream
ϵ: maximal approximation error

$T \leftarrow \emptyset$ // Initially empty set
$N \leftarrow 1$ // Number of observed events
$w \leftarrow \lceil \frac{1}{\epsilon} \rceil$ // Bucket width
forever do
  $e \leftarrow observe(S)$
  $b_{curr} \leftarrow \lceil \frac{N}{w} \rceil$
  /* Is there a tuple in T with e as first component? */
  if e *is already in* T **then**
    Increment the frequency of e in T
  else
    Insert $(e, 1, b_{curr} - 1)$ in T
  end
  if $N \mod w = 0$ **then**
    forall the $(a, f, \Delta) \in T$ *s.t.* $f + \Delta \leq b_{curr}$ **do**
      Remove (a, f, Δ) from T
    end
  end
  $N \leftarrow N + 1$
end

then the new event is inserted. Periodically, the system converts the memory into a standard event log, and classical process mining algorithms are applied on it. The literature, in Babcock et al. (2002), identifies at least two memory models capable of storing event: *sequence based* and *timestamp based*. The first approach (which is typically implemented with *sliding windows*) consists of a FIFO queue of fixed size. The observed events are stored in the window, and once the maximum capacity is reached, the oldest event is removed. In a timestamp-based window model, the approach is very similar, but the memory size is not fixed. Instead, the removal is based on the "age" of the observation: the memory keeps only the events observed within the given time span.

This approach comes with several advantages, such as the possibility to reuse every mining algorithm already available for event logs. However, the memory management is extremely problematic and has a huge impact on the performance of the approach. Specifically, all window-based approaches have poor summarizing capabilities since, for example, duplicate events require two "memory slots," even though they may not provide new information.

Problem reduction. The second possible way of tackling the streaming process mining problems is to employ a task-based technique. For example, it is possible to *reduce* the process mining problem to another well-established problem and therefore reuse algorithms specifically devised and optimized for the necessity at hand. Of course, it is also possible to devise a completely new algorithm specifically tailored to solve the given situation. This approach, for example, has been used to reduce the problem of streaming process discovery to the *frequency counting problem*. This problem consists of counting the frequencies of given variables over a stream. In order to reduce the process discovery to such problem, it is important to understand *what is* a variable in the process mining context and whether it is possible to identify it.

An example of efficient algorithm for the approximated frequency counting problem is Lossy Counting, here summarized in Algorithm 2 and described in Manku and Motwani (2002). The idea is to conceptually split the observed stream in buckets, each with a fixed size. The approach takes as input the stream and the maximal approximation error on the counting $\epsilon \in [0, 1]$, which drives the size of each bucket. The approach stores entries in a data structure T where each

component (e, f, Δ) provides the element e of the stream, the estimated frequency for it f, and the maximum number of times it could have occurred Δ. When a new event is observed in the stream, the algorithm checks if it is already in T and, in case, it increments its counter f by one. Otherwise a new entry in T is created. Periodically (i.e., every time a new conceptual bucket starts), the algorithm cleans the memory, by removing elements not frequently observed. This algorithm has no memory bound. Specifically, the size of the data structure T depends on the stream and on the approximation error. However, a variation of the algorithm to enforce fixed amount of memory is described in Da San Martino et al. (2012).

The most relevant benefit of the problem reduction approach is that, once the process mining problem has been reduced to the new one, it is possible to use algorithms already devised for the reduced problem. However, such reduction might not be trivial and all assumptions of the used approach have to be met.

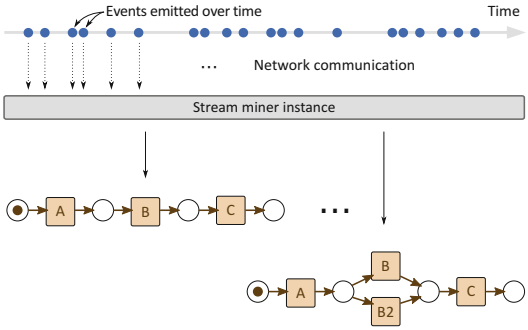
Offline computation. The last approach we present consists of moving the computation of the solutions to the given problem from online setting to offline. In other words, the idea is to identify and solve all subproblems the stream may provide. Adopting this approach will help us in dealing with all situations we are going to observe, still keeping the complexity of the online processing constant.

The advantage of this approach is the possibility of having extremely expensive solutions "cached" in advance which are then just reused whenever needed. Though there are several drawbacks, it is not possible to apply this approach to all online process mining problems. Additionally, by computing everything in advance, we lose the possibility of adapting the precomputed solutions to the contextual information, which might be uniquely specific to the running process instance.

In the upcoming subsections, one example of each technique will be presented and detailed by presenting two process mining activities.

Process Discovery

The first problem we present is the online process discovery. A graphical conceptualization of the problem is reported in Fig. 2. The idea is to have a source which is generating an event stream. This event stream is consumed by a miner which generates a representation of the underlying process model and keeps it up-to-date with respect to the observed behavior.



Streaming Process Discovery and Conformance Checking, Fig. 2 Conceptualization of the streaming process discovery idea as in Burattin et al. (2014b)

The first approach available to tackle this problem is reported in Burattin et al. (2012, 2014b). In their works, authors employ a problem reduction strategy. Specifically, they reduce the Heuristics Miner algorithm, described in van der Aalst and Weijters (2003), to the frequency counting problem. To do that, they assume *direct following relationships* as variables, and by counting them, they are able to use the Heuristics Miner's metric to reconstruct the high-level business patterns in terms of Heuristics Net or Petri Net. Authors test different algorithms to solve the frequency counting problem, such as Lossy Counting and Lossy Counting with Budget. As baseline approach, authors use a sliding window mechanism where Heuristics Miner is iteratively applied. The sliding window approach is constantly outperformed by other approaches which provide a better usage of the available resources. A similar approach, called StrProM, tracks the direct following relationships. This approach, presented in Hassani et al. (2015), keeps an updated prefix tree by deriving a solution based on Lossy Counting with Budget. The direct following relationships are then used to construct a process model using the set of rules of Heuristics Miner.

As for the previously mentioned approaches, in Maggi et al. (2013) and Burattin et al. (2014a, 2015), authors investigate the problem of discovering a process represented in Declare (cf. Pesic et al. 2007). Specifically, their idea is to instantiate several "replayers," one for each Declare template to mine. Then, specific behavior for each template is implemented. To keep track of the replay statuses, authors use Lossy Counting-based strategies. Authors apply sliding window as baseline, and again, the performances of the Lossy Counting strategies are better with respect to the simple application of offline approaches over a sliding window.

In Redlich et al. (2014b), authors present the adaptation of CCM to cope with event streams. The basic idea of CCM (cf. Redlich et al. 2014a) is to identify subsequences of events in order to identify footprints of specific process patterns. In CCM, relevant patters and corresponding footprints are described. Aging factors are employed to the collected information in order to give more importance to recent behavior.

The most recent work tackling the online process discovery is reported in van Zelst et al. (2017b). In their paper, authors generalize previously instantiated concepts: they present an architecture, namely, S-BAR, which keeps an updated abstract representation of the stream (e.g., direct follow relationships), and they use it as starting point to infer an actual process model. In their work, authors present the adaptations of different mining algorithms: α (cf. van der Aalst et al. 2004), Heuristics Miner (cf. van der Aalst and Weijters 2003), and Inductive Miner (cf. Leemans et al. 2013). To show the generability of their abstract representation, authors also show a miner based on region theory (cf. van der Aalst et al. 2008). In order to keep their abstraction updated, authors reduce their problem to frequency counting, thus using Lossy Counting, Space Saving (cf. Metwally et al. 2005), and Frequent (cf. Karp et al. 2003).

Conformance Checking

The problem of online conformance checking has received attention in the declarative domain. In this case, it used to be called *operational support*, and its aim is to understand whether a set of constraints is being violated or not. To achieve that, in Maggi et al. (2011, 2012), authors devise an approach to represent the behavior as an automaton and executions are replayed on it. Additionally, each process instance is labeled with one of four possible fulfillment states: permanently/temporarily violated/fulfilled.

Concerning imperative models, online conformance checking received less attention. At present time, only two approaches have been specifically designed to tackle the online conformance checking problem. One approach, described in van Zelst et al. (2017a), aims at reusing the concept of *alignment* in order to compute the optimal alignment just for the prefix of the trace seen up to a given point in time. To this end, authors first devise a technique to compute prefix alignments. Authors prove the optimality of the prefix alignment they discover. Up to this point, their approach is incremental but not really online (i.e., it is able to work on partial cumulative information, but in theory they need infinite memory to backtrack in order to find the optimal alignment). To solve this issue, authors mention the possibility to implement memory management, either falling into a window-based model or as problem reduction. Authors, however, do not implement or test either memory management approach. However, they test their technique against different number of backtracking steps required to find the prefix alignment.

Another conformance checking approach belongs to the offline computation category and is described in Burattin and Carmona (2017) and Burattin (2017). In this case, given a Petri Net as input, authors describe a technique to elicit a transition system containing all possible transitions from one marking to another one, even those which are not described by the original model. Each transition in this elicited model is then associated with a cost. Transitions allowed by the original Petri Net have cost 0; all others have cost > 0. This way, by replaying a trace on such transition system and summing the costs, authors show that conformant traces will have cost 0 and non-conformant trace always have cost larger than 0. Additionally, authors ensure the

determinism of such transition system and the possibility, from each state, to have one transition for each possible activity. These two last properties guarantee the suitability of the approach for online settings.

Another conformance checking approach is presented in Weber et al. (2015). In this case, authors propose a RESTful service which performs a token replay on a BPMN model. In particular, authors implemented a *token pull mechanism*. Authors refer this approach as online primarily because of its interface, while no explicit guarantee is mentioned in terms of memory usage and computational complexity.

Other Applications

Online process mining has been applied also to discover cooperative structures out of event streams. For example, in van Zelst et al. (2016), authors are able to process an event stream and update the set of relationships of a cooperative resource network.

Additionally, in order to conduct research on stream process mining, it is useful to simulate an event stream for which we know the actual original source. To achieve that, mainly two approaches are available. The first is a tool called PLG2 (http://plg.processmining.it/), described in Burattin (2016). It allows to generate a random model or to load a BPMN file and stream events referring to actual executions. The stream is sent over a TCP/IP connection, and the tool allows different configurations in terms of noise and concept drift. The second tool is described in van Zelst et al. (2015). This tool allows researchers to design a model – as Petri Net – in CPN (http://www.cpntools.org/) and then import it into ProM (http://www.promtools.org/) where the `XESEventStream Publisher` plugin can be used to simulate a stream. Please note that, in this case, the stream exists just within ProM.

Key Applications

Due to the novelty of the topic, we are not aware of any deployment of streaming process mining techniques in real settings. However, more generally, online process mining techniques are relevant in all cases where it is important to have results in real time, to immediately enact proper responses. In this section, examples related to IT setting will be provided, but business-oriented applications are absolutely possible and relevant as well.

Online process discovery might be useful in settings where it is important to analyze immediately the behavior of the system. For example, reconstructing the behavior of the services used in a website might be useful in order to see what is currently under stress and what is going to be used afterward. Exploiting this information could improve the resource allocation (e.g., upscaling or downscaling the server capacity on the fly).

Online conformance checking is also useful whenever it is important to immediately detect deviations from reference behavior to enact proper countermeasures. For example, the kernel of an operating system exposes some services for applications. These services should be combined in some specific ways (e.g., a file should be `open()` and then either `write()` or `read()` or both appear, and eventually the file should be `close()`) which represent the reference behavior. If an application is observed strongly violating such behavior, it might be an indication of strange activities going on, for example, in order to bypass some imposed limitations or privileges.

Future Directions for Research

As previously mentioned, online process mining is a relatively new area of investigation. Some techniques are available for the discovery of the process (both as imperative and declarative models). Very recently, online conformance checking also received attention, but several improvements are still needed.

First of all, techniques should improve their efficiency, for example, in terms of memory consumption. This represents an important research direction since, in the online setting, the amount of available memory is fixed, thus representing a

key resource. To tackle this problem, it is necessary to work on the summarization capabilities used by the algorithms in order to find more compact ways of storing the same information.

Related to the previous point is the quality and quantity of information, and contextual data algorithms are able to extract out of the same event stream. For example, a process discovery algorithm might be extended to extract more complex control flow patterns, or the algorithm might be modified to return not just the result but the confidence on the provided outcomes.

Additionally, there are more technical issues that algorithms should be able to cope with, for example, dealing with a stream where the arrival time of events does not coincide with their actual execution. In this case, it would be necessary to reorder the list of events belonging to the same process instance before mining them. Please note that assuming different orders implies a sort of different element models of the stream (i.e., it becomes an "insert-delete stream model," where the order of events can change). Another relevant issue might be the inference of the termination of a process instance.

Cross-References

- Automated Process Discovery
- Conformance Checking
- Declarative Process Mining
- Definition of Data Streams
- Introduction to Stream Processing Algorithms

References

Aggarwal CC (2007) Data streams: models and algorithms. Advances in database systems. Springer, Boston. https://doi.org/10.1007/978-0-387-47534-9

Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems, pp 1–16. https://doi.org/10.1145/543614.543615

Bifet A, Kirkby R (2009) Data stream mining: a practical approach. Technical report, Centre for open software innovation – The University of Waikato

Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: massive online analysis learning examples. J Mach Learn Res 11:1601–1604

Burattin A (2016) PLG2 : Multiperspective process randomization with online and offline simulations. In: Online proceedings of the BPM Demo Track 2016, CEUR-WS.org, vol 1789, pp 1–6

Burattin A (2017) Online conformance checking for petri nets and event streams. In: CEUR Workshop Proceedings, vol 1920

Burattin A, Carmona J (2017, in press) A framework for online conformance checking. In: Proceedings of the 13th international workshop on business process intelligence (BPI 2017). Springer

Burattin A, Sperduti A, van der Aalst WM (2012) Heuristics miners for streaming event data. ArXiv CoRR http://arxiv.org/abs/1212.6383

Burattin A, Maggi FM, Cimitile M (2014a) Lights, camera, action! business process movies for online process discovery. In: Proceedings of the 3rd international workshop on theory and applications of process visualization (TAProViz 2014)

Burattin A, Sperduti A, van der Aalst WM (2014b) Control-flow discovery from event streams. In: Proceedings of the IEEE congress on evolutionary computation. IEEE, pp 2420–2427. https://doi.org/10.1109/CEC.2014.6900341

Burattin A, Cimitile M, Maggi FM, Sperduti A (2015) Online discovery of declarative process models from event streams. IEEE Trans Serv Comput 8(6):833–846. https://doi.org/10.1109/TSC.2015.2459703

Da San Martino G, Navarin N, Sperduti A (2012) A lossy counting based approach for learning on streams of graphs on a budget. In: Proceedings of the twenty-third international joint conference on artificial intelligence. AAAI Press, pp 1294–13010

Dumas M, La Rosa M, Mendling J, Reijers HA (2013) Fundamentals of business process management. Springer

Gaber MM, Zaslavsky A, Krishnaswamy S (2005) Mining data streams: a review. ACM Sigmod Rec 34(2):18–26. https://doi.org/10.1.1.80.798

Gama J (2010) Knowledge discovery from data streams. Chapman and Hall/CRC, Boca Raton. https://doi.org/10.1201/EBK1439826119

Golab L, Özsu MT (2003) Issues in data stream management. ACM SIGMOD Rec 32(2):5–14. https://doi.org/10.1145/776985.776986

Hassani M, Siccha S, Richter F, Seidl T (2015) Efficient process discovery from event streams using sequential pattern mining. In: 2015 IEEE symposium series on computational intelligence, pp 1366–1373. https://doi.org/10.1109/SSCI.2015.195

Karp RM, Shenker S, Papadimitriou CH (2003) A simple algorithm for finding frequent elements in streams and bags. ACM Trans Database Syst 28(1):51–55. https://doi.org/10.1145/762471.762473

Leemans SJJ, Fahland D, van der Aalst WM (2013) Discovering block-structured process models from event logs – a constructive approach. In: Proceedings of Petri

nets. Springer, Berlin/Heidelberg, pp 311–329. https://doi.org/10.1007/978-3-642-38697-8_17
Maggi FM, Montali M, Westergaard M, van der Aalst WM (2011) Monitoring business constraints with linear temporal logic: an approach based on colored automata. In: Proceedings of the 9th international conference on business process management. Springer, Berlin/Heidelberg, pp 132–147. https://doi.org/10.1007/978-3-642-23059-2_13
Maggi FM, Montali M, van der Aalst WM (2012) An operational decision support framework for monitoring business constraints. In: Proceedings of 15th international conference on fundamental approaches to software engineering (FASE), pp 146–162. https://doi.org/10.1007/978-3-642-28872-2_11
Maggi FM, Bose RPJC, van der Aalst WM (2013) A knowledge-based integrated approach for discovering and repairing declare maps. In: 25th international conference, CAiSE 2013, 17–21 June 2013. Springer, Berlin/Heidelberg/Valencia, pp 433–448. https://doi.org/10.1007/978-3-642-38709-8_28
Manku GS, Motwani R (2002) Approximate frequency counts over data streams. In: Proceedings of international conference on very large data bases. Morgan Kaufmann, Hong Kong, pp 346–357
Metwally A, Agrawal D, Abbadi AE (2005) Efficient computation of frequent and Top-k elements in data streams. In: Database theory – ICDT 2005. Springer, Berlin/Heidelberg, pp 398–412. https://doi.org/10.1007/978-3-540-30570-5_27
Pesic M, Schonenberg H, van der Aalst WM (2007) DECLARE: full support for loosely-structured processes. In: Proceedings of EDOC. IEEE, pp 287–298. https://doi.org/10.1109/EDOC.2007.14
Redlich D, Molka T, Gilani W, Blair G, Rashid A (2014a) Constructs competition miner: process control-flow discovery of BP-domain constructs. In: Proceedings of BPM 2014, pp 134–150. https://doi.org/10.1007/978-3-319-10172-9_9
Redlich D, Molka T, Gilani W, Blair G, Rashid A (2014b) Scalable dynamic business process discovery with the constructs competition miner. In: Proceedings of the 4th international symposium on data-driven process discovery and analysis (SIMPDA 2014), vol 1293, pp 91–107
van der Aalst WM, Weijters TAJMM (2003) Rediscovering workflow models from event-based data using little thumb. Integr Comput Aided Eng 10(2):151–162
van der Aalst WM, Weijters TAJMM, Maruster L (2004) Workflow mining: discovering process models from event logs. IEEE Trans Knowl Data Eng 16:2004
van der Aalst WM, Günther CW, Rubin V, Verbeek EHMW, Kindler E, van Dongen B (2008) Process mining: a two-step approach to balance between underfitting and overfitting. Softw Syst Model 9(1):87–111. https://doi.org/10.1007/s10270-008-0106-z
van Zelst SJ, van Dongen B, van der Aalst WM (2015) Know What you stream: generating event streams from CPN models in ProM 6. In: CEUR workshop proceedings, pp 85–89
van Zelst SJ, van Dongen B, van der Aalst WM (2016) Online discovery of cooperative structures in business processes. In: Proceedings of the OTM 2016 conferences. Springer, pp 210–228
van Zelst SJ, Bolt A, Hassani M, van Dongen B, van der Aalst WM (2017a) Online conformance checking: relating event streams to process models using prefix-alignments. Int J Data Sci Analy. https://doi.org/10.1007/s41060-017-0078-6
van Zelst SJ, van Dongen B, van der Aalst WM (2017b) Event stream-based process discovery using abstract representations. Knowl Inform Syst pp 1–29. https://doi.org/10.1007/s10115-017-1060-2
Weber I, Rogge-Solti A, Li C, Mendling J (2015) CCaaS: online conformance checking as a service. In: Proceedings of the BPM demo session 2015, vol 1418, pp 45–49
Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23(1):69–101. https://doi.org/10.1007/BF00116900

Streaming SQL Queries

▸ Continuous Queries

StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing

André Martin[1], Andrey Brito[2], and Christof Fetzer[1]
[1]TU Dresden, Dresden, Germany
[2]UFCG, Campina Grande, Brazil

Introduction

During the past decade, we have been witnessing a massive growth of data. In particular the advent of new mobile devices such as smartphones, tablets and online services like Facebook and Twitter created a complete new era for data processing. Although there exist already well-established approaches such as MapReduce (Dean and Ghemawat, 2008) and its open-source implementation Hadoop (2015) in order to cope with these large amounts of data, there is a recent trend of moving

away from batch processing to low-latency online processing using event stream processing (ESP) systems. Inspired by the simplicity of the MapReduce programming paradigm, a number of open-source as well as commercial ESP systems have evolved over the past years such as Apache S4 (Neumeyer et al., 2010; Apache, 2015) (originally pushed by Yahoo!), Apache Storm (2015) (Twitter), and Apache Samza (2015) (LinkedIn), addressing the strong need for data processing in near real time.

Since the amount of data being processed often exceeds the processing power of a single machine, ESP systems are often carried out as *distributed systems* where multiple nodes perform data processing in a cooperative manner. However, with an increasing number of nodes used, the probability for a failure increases which can lead either to partial or even full system outages. Although several well-established approaches to cope with system failures for distributed systems are known in literature, providing fault tolerance for ESP systems is challenging as those systems operate on constantly flowing data where the input stream cannot be simply stopped and restarted during system recovery.

One possibility for providing fault tolerance in ESP systems is the usage of checkpointing and logging, also known as *rollback recovery/-passive replication* in literature where the state of an operator is periodically checkpointed to some fault-tolerant stable storage and so-called in-flight events are kept in in-memory logs at upstream nodes (upstream backup) (Hwang et al., 2005; Gu et al., 2009). During system recovery, the most recent checkpoint is being loaded, and previously in-flight events are replayed. An alternative to rollback recovery is *active replication* (Schneider, 1990; Martin et al., 2011a) where two identical copies of an operator are deployed on different nodes performing redundant processing. If one of the two copies crashes, the system continues to operate without having to initiate a long recovery procedure as in passive replication.

Although active replication provides the quickest recovery, it requires almost twice the resources, while passive replication consumes only little resources; however, it suffers from long recovery times. Despite the fact that both fault tolerance approaches have different characteristics with regard to recovery times and resource overhead, both require a *deterministic* processing of events. Deterministic processing ensures that all replicas process events in the same predefined order which is important in order (i) to reliably filter out duplicated events when using active replication and (ii) to provide reproducibility of events needed in order to recover precisely (i.e., neither loosing any events nor processing events twice) using passive replication. However, the use of deterministic execution imposes a non-negligible overhead as it increases processing latency and lowers throughput at the same time due to the cost of event ordering.

Key Research Findings

This chapter presents three approaches to reduce the overhead imposed by fault tolerance in ESP systems. They are presented in the following order: First,

1. the architecture and implementation of StreamMine3G are discussed, an ESP system that was built from scratch to study and evaluate novel fault tolerance and elasticity mechanisms,
2. an algorithm to reduce the overhead imposed by deterministic execution targeting commutative tumbling windowed operators and improving the throughput by several orders of magnitude when used with passive and active replication,
3. an approach to improve the overall system availability by utilizing spare but paid cloud resources, and
4. an adaption-based approach that minimizes the operational costs by selecting the least expensive fault tolerance scheme at runtime based on user-provided constraints.

Roadmap section "StreamMine3G Approach" introduces the reader to the architecture and implementation of StreamMine3G, the ESP system used for evaluating the proposed approaches. In section "Lowering Runtime Overhead for Pas-

sive Replication", an approach is presented that lowers the overhead of event ordering by introducing the notion of an *epoch* and evaluated it for the use with passive replication. An extension of this approach is discussed in section "Lowering Runtime Overhead for Active Replication" for the use with active replication by proposing an epoch-based merge algorithm and a lightweight consensus protocol. Next, section "Improving Resource Utilization and Availability Through Active Replication" details how to improve system availability by using a hybrid approach of passive standby and active replication by utilizing spare but paid cloud resources, while in section "Adaptive and Low-Cost Fault Tolerance for Cloud Environments", an adaptation approach for fault tolerance is presented that allows to lower the overall resource consumption while still satisfying user-specified constraints such as recovery time and recovery semantics. Finally, the chapter concludes in section "Conclusions" with a short summary of the approaches and the contributions.

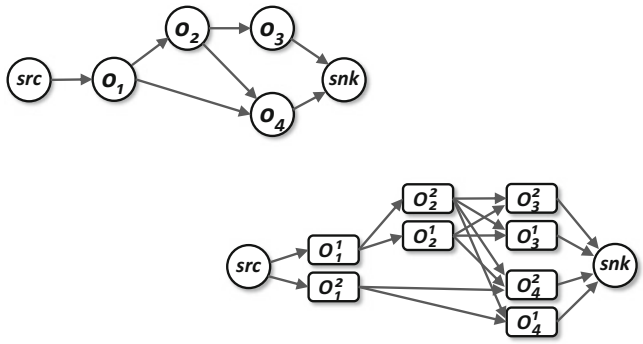
StreamMine3G Approach

The following section provides a brief overview about StreamMine3G, the ESP system used to implement and evaluate the proposed approaches.

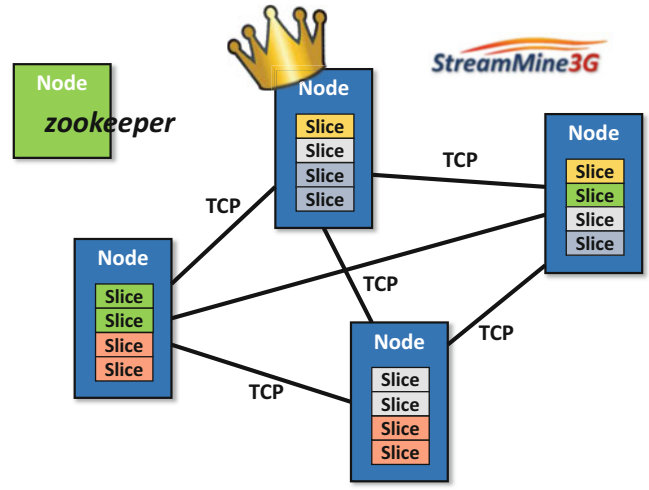
StreamMine3G is a highly scalable ESP system targeting low-latency data processing of streaming data. In order to analyze data, users can either opt for writing their own custom operators using the provided MapReduce-like interface and implementing a user-defined function (UDF) or choose from an existing set of standard complex event processing (CEP) operators such as filter, join, aggregation, and others. In addition to the operators, users must specify the order events are supposed to traverse the previously selected operators using a topology. A topology in StreamMine3G is represented by a directed acyclic graph (DAG) where each vertex, i.e., an operator, can have multiple upstream and downstream operators as shown in Fig. 1 (left). In order to achieve scalability, operators in StreamMine3G are partitioned as depicted in Fig. 1 (right). Each partition processes only a subset of events from the incoming data stream. For data partitioning, users can either implement their own custom partitioner as in MapReduce or use the provided hash-based or key-range-based partitioner.

A typical StreamMine3G cluster consists of several nodes where each node runs a single StreamMine3G process hosting an arbitrary number of operator partitions, named slices as shown in Fig. 2. One of such nodes takes up the role of a manager which is responsible for placing operator partitions across the set of available nodes as well as moving partitions (through a migration mechanism) to other nodes for load balancing in situations of overload or underutilization. An overload can be detected by the manager node by analyzing the system utilization of each node, which is periodically reported through heartbeat messages exchanged between nodes.

In order to prevent the node hosting the manager component being the single point of failure, the state of the component is stored in Zookeeper (Hunt et al., 2010) upon each



StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing, Fig. 1 Logical (top left) and physical (bottom right) representation (with two partitions for each operator) of a query graph



StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing, Fig. 2 A STREAMMINE3G cluster. One node acts as a master/manager, while others serve as workers hosting an arbitrary number of slices (operator partitions)

reconfiguration of the system. In the event of a crash of the node, another node can transparently take over the role of the manager by simply recovering with the previously persisted state.

Lastly, STREAMMINE3G supports the implementation of stateless and stateful operators. However, contrary to other ESP systems such as Apache S4 and Storm that have either no, or only limited, state support, STREAMMINE3G offers an explicit state management interface to its users. The interface frees the users from the burden of having to implement their own bridle locking mechanism to ensure consistent state modifications when processing events concurrently (to exploit multiple cores) and provides a full stack of mechanisms for state checkpoints, recovery, and operator migration. In order to use these mechanisms, users are only required to implement appropriate methods for serialization and de-serialization of the state that can comprise arbitrary data structures.

Lowering Runtime Overhead for Passive Replication

In the following section, an approach for lowering the overhead for passive replication by introducing the notion of an epoch is presented.

Introduction and Motivation

ESP applications are often long-running operations that continuously analyze data in order to carry out some form of service. In order to identify patterns and correlations between consecutive events, operators are often implemented as stateful components. One way for providing fault tolerance for such components is to combine periodic checkpointing with event logging for a later replay. However, since events may arrive in different orders at an operator during a recovery than they would have arrived originally due to small delays in network packet delivery, the immediate processing of such events would lead to possibly inconsistent results. One way of preventing such situations is to *order events* prior to the processing in order to ensure a deterministic input to the operator code at all times. However, the ordering of events is costly as it introduces latency and lowers throughput as shown later.

Fortunately, many ESP operators used in ESP applications share the property of *commutativity* and operate on jumping windows where the order of processing within such windows is irrelevant for the computation of the correct result. Examples for such operators are joins and aggregations. However, processing the same event twice or even missing an event may still distort the result of those operators. Hence, determinism is still needed in order to provide *exactly once processing semantics*.

Approach

Based on the observation that many operators are commutative and operate on jumping windows, the notion of an *epoch* is introduced. An epoch comprises a set of events based on their timestamps and matches the length of the window an operator is working on. In order to assign events correctly to those epochs, i.e., the time-based windows, events are equipped with monotonically increasing timestamps. The key idea of the approach is to process events within such epochs, i.e., windows in *arbitrary order*, however, *processing epochs itself in order*. Exactly once semantics can now be provided by solely performing checkpointing and recovery at *epoch boundaries* as at those points in time the system still provides a deterministic snapshot.

Evaluation and Results

For the evaluation of the approach, a canonical streaming word count application has been implemented that consists of two operators: a stateless map operator that splits lines read from a Wikipedia corpus file into individual words which are then accumulated by a stateful reduce operator. The stateful operator summarizes the word frequencies using a jumping window, i.e., an epoch where the length of the epoch is defined in terms of file position the word originated from in the source file. The performance of the epoch-based approach has been evaluated by comparing it with an execution that does not perform any event ordering and an execution that performs a strict event ordering. In strict ordering, every single event is ordered rather than applying the weak ordering scheme based on epochs. The results of the measurements are shown in Fig. 3.

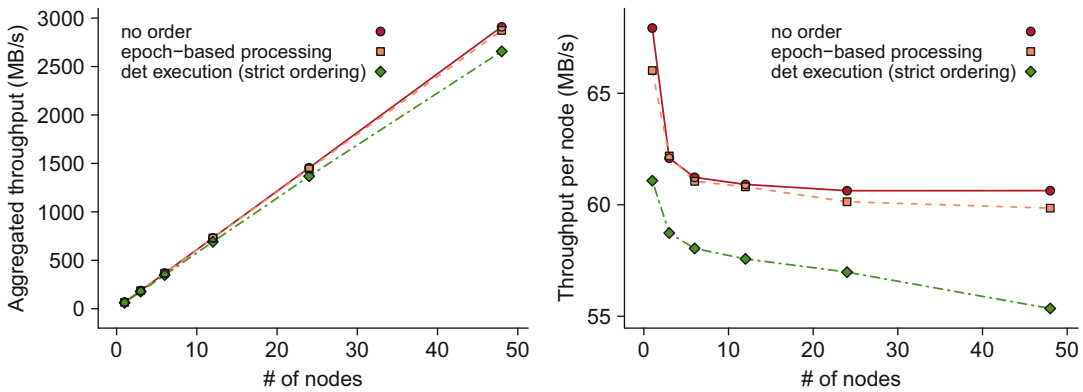
Figure 3 (left) shows the accumulated throughput for the experiment running on a 50-node cluster, while Fig. 3 (right) depicts the per node throughput. The experiments reveal that using the proposed weak ordering scheme, a similar throughput as when not applying any ordering can be achieved, however, providing precise recovery semantics as when using strict ordering, i.e., deterministic execution.

Lowering Runtime Overhead for Active Replication

In the previous section, an approach for reducing the overhead of event ordering was presented that provides *exactly once processing semantics* and *precise recovery* when used with passive replication which is traditionally based on checkpointing and in-memory event logging. In this section, an extension will be presented that can be used with active replication.

Introduction and Motivation

Active replication is a useful approach to recover applications that accumulate large portions of state as the secondary instance is holding the



StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing, Fig. 3 Aggregated (left) and per node throughput (right) with increasing number of nodes. The epoch-based approach significantly outperforms deterministic execution

state already in memory rather than reading it from disk during a recovery. However, in order to use active replication, a costly atomic broadcast or deterministic execution (i.e., strict event ordering) is needed in order to ensure consistent processing across all replicas. However, when using commutative and windowed operators, event ordering solely serves the purpose of maintaining consistency across replicas but does not have any impact on correctness due to the commutativity.

Approach

Inspired by the previous epoch-based processing approach, the following approach can be considered as an extension as it performs an *epoch-based deterministic merge* that ensures correctness for active replication, however, at a much lower cost than a strict event order/merge. The key idea of the approach is to merge epochs rather than individual events which is far less costly than a strict per event merge as shown in the evaluation below.

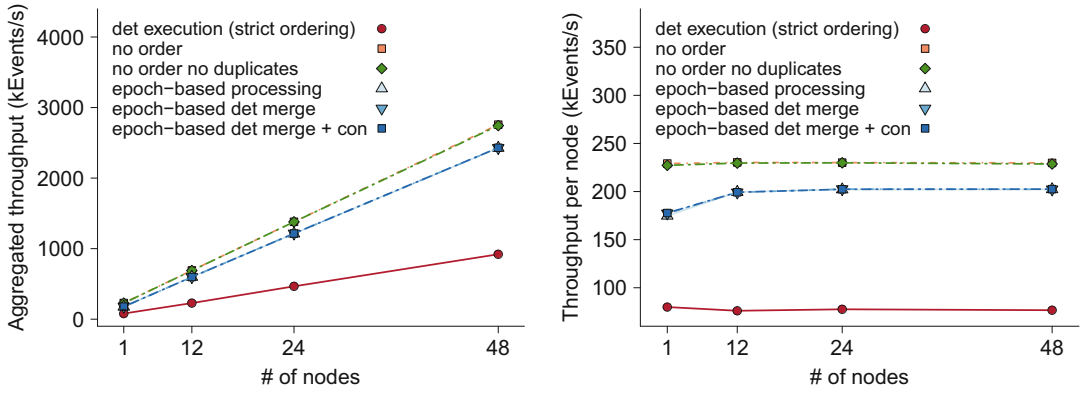
In order to perform an epoch-based deterministic merge, events arriving from different upstream channels are enqueued on a per epoch basis in separate so-called *epoch bags* first. Once an epoch completes, i.e., all channels have passed the epoch boundary, the content of the epoch bags is merged by processing the bags in the order of predefined channel identifiers. Since the channel identifiers are globally defined and events from upstream operators are delivered in FIFO order through TCP, the final sequence of events is identical and deterministic for all replicas.

In case upstream operator replicas or sources deliver identical but differently ordered sequences of events and in order to reduce latency caused by stragglers, a *lightweight consensus protocol* can be used that selects for the available upstream replicas the set of bags to merge so that all downstream peers process the same sets of events. The protocol stops furthermore the propagation of non-determinism while decreasing latency at the same time.

Evaluation and Results

For the experimental evaluation of the approach, a canonical application operating on jumping windows was implemented that consists of three operators, a partitioned source operator, an equi-join that combines the output from the source operator, and a sink. In order to assess the performance of the approach, it was compared with an out-of-order execution, a strict ordering, the epoch-based merge, and the consensus-based protocol. The outcome of the experiments is depicted in Fig. 4.

Figure 4 (left) depicts the accumulated throughput for the experiment running on a 50-node cluster, while Fig. 4 (right) depicts the per node throughput. As shown in the figures, the epoch-based deterministic merge has a noticeable higher throughput than strict determinism (ordering), while there is only a



StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing, Fig. 4 Horizontal scaling – aggregated (left) and per node (right) throughput with increasing number of nodes

marginal difference for the consensus-based variant in comparison to the epoch-based deterministic merge without agreement.

Improving Resource Utilization and Availability Through Active Replication

While the objective of the previously presented approaches was to minimize the runtime overhead for fault tolerance by introducing a weak ordering scheme based on the notion of epochs, the approach presented next improves system availability by efficiently using spare but paid resources in commonly available cloud environments.

Introduction and Motivation

ESP systems are naturally highly dynamic systems as they process data often originating from live data sources such as Twitter or Facebook where the streams have highly varying data rates that may change by several orders of magnitude within relative short periods of time. In order to cope with those peak loads and to prevent the unresponsiveness of the system, the systems are usually run at conservative utilization levels often as low as 50%. Although the cloud computing model enables customers to acquire resources quite easily, migration and rebalancing mechanisms are still not fast enough to accommodate sudden load spikes forcing the service providers to run their applications at low utilization levels. However, cloud users are nevertheless charged by full hours regardless of the actual resource consumption of their virtual machines.

Approach

In order to fully utilize all available resources a virtual machine offers, a *hybrid approach* is used for fault tolerance where a transition between *active replication* and *passive standby* based on the utilization of the system is made. In order to transition between the two states, a *priority scheduler* is used that prioritizes the processing of the primary and transparently pauses secondaries once resources are exhausted. Hence, the system transparently transitions between active replication and passive standby where the secondary is paused until sufficient resources become available again. In order to keep the secondary up-to-date during high system utilization, the state of the primary is periodically checkpointed to the memory of the secondary which allows the secondary to prune enqueued but not yet processed events from queues. Using an interleaved partitioning scheme for the placement of primary and secondaries across the cluster, the overhead imposed on nodes during system recovery can furthermore be decreased.

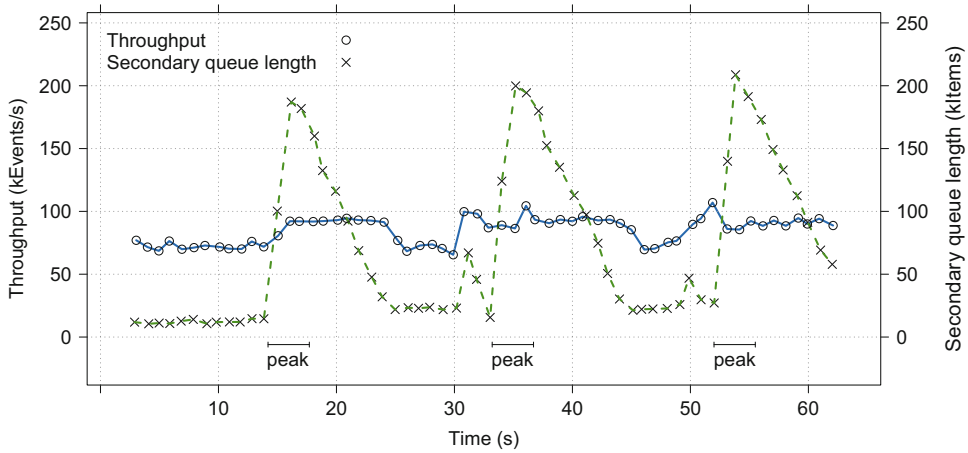
Evaluation and Results

In the following experiment, the system behavior of the proposed solution has been investigated in the event of load peaks. To simulate spikes, a load generator to emit events at different rates for predefined periods of time was used. Figure 5 depicts the aggregated throughput for a single node and the status of the input queues of secondary slices on that node over time. In this experiment, the load generator introduced load spikes every 20 s for 2 s.

During a load peak, no events at the secondary slices on that node are being processed; hence queues grow quickly. Once the load decreases, secondaries resume the processing of events; hence, the amount of events in the queues of the secondary slices shrink. Note that the aggregated throughput of the node remains high until the shrinking process has been fully completed. During the spike, the aggregated throughput was higher due to the increase in load on the primary slices; after the spike, the throughput is higher due to the accumulated load on the secondary slices.

Adaptive and Low-Cost Fault Tolerance for Cloud Environments

In the previous section, an approach to utilize spare but already paid resources was presented that improves system availability by dynamically transitioning between active replication and passive standby during runtime. The following



StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing, Fig. 5 Event throughput and queue length behavior of secondary slice queues with induced load spikes

section presents an approach that lowers the overall resource consumption by selecting the fault tolerance scheme at runtime that consumes the least amount of resources while still guaranteeing user-defined constraints such as *recovery time* and *recovery semantics*.

Introduction and Motivation

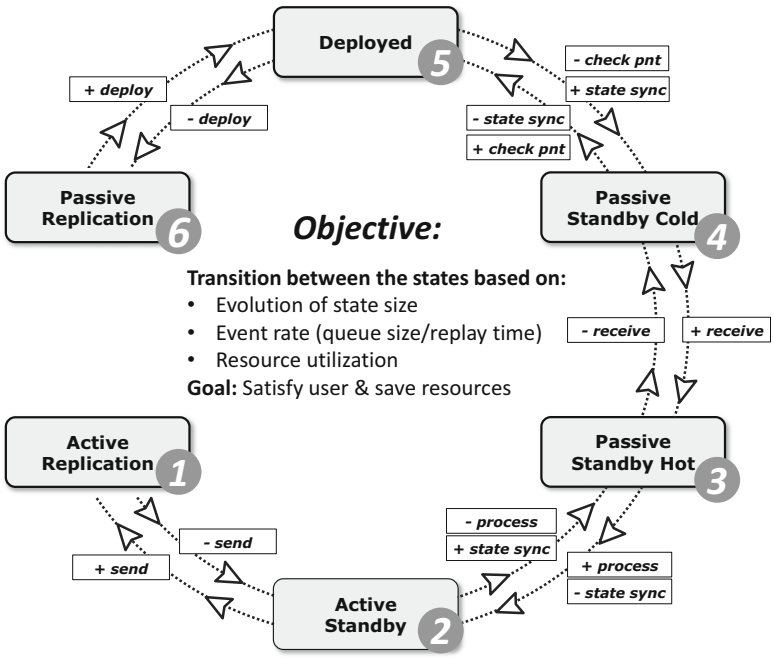
Fault tolerance in ESP systems can be carried out through various mechanism and replication schemes. For example, in *active replication*, redundant processing is used as a mechanism to carry out fault tolerance where as in *passive replication*, a combination of periodic checkpointing and event logging is used in order to mitigate system crashes. Although active replication provides a quick recovery, it comes with a high price as it consumes almost twice of the resources, while passive replication consumes only little resources; however, it suffers from a long recovery time. Besides active and passive replication, there exist several more schemes to carry out fault tolerance such as *active* and *passive standby* where recovery time is traded by resource usage.

Choosing the right fault tolerance scheme is not trivial as all those schemes have different resource footprints and recovery times. Moreover, the footprint and recovery time for those schemes are not static as they strongly depend on system parameters that can greatly vary during the course of processing. For example, the recovery time for passive replication strongly depends on the size of the checkpoint that must be read during a recovery. However, the size of a checkpoint strongly depends on the incoming data rate when considering a stateful sliding window operator.

Approach

In order to free the user from the burden of choosing an appropriate fault tolerance scheme for his application, a *fault tolerance controller* that takes decisions on behalf of the user at runtime can be utilized. The controller takes into account user-provided constraints such as the desired *recovery time* and *recovery semantics*, i.e., precise or gap recovery.

Therefore, StreamMine3G was extended to support six different fault tolerance schemes the controller can choose from as shown in Fig. 6: ① *active replication*, ② *active standby*, ③ *passive standby hot* and ④ *cold*, ⑤ *deployed*, and ⑥ *passive replication*. The schemes have different characteristics with regard to resource consumption of CPU, memory, network bandwidth, the amount of nodes used, and recovery time. In order to choose the correct scheme, the controller uses an estimation-based approach where historically collected measurements are continuously



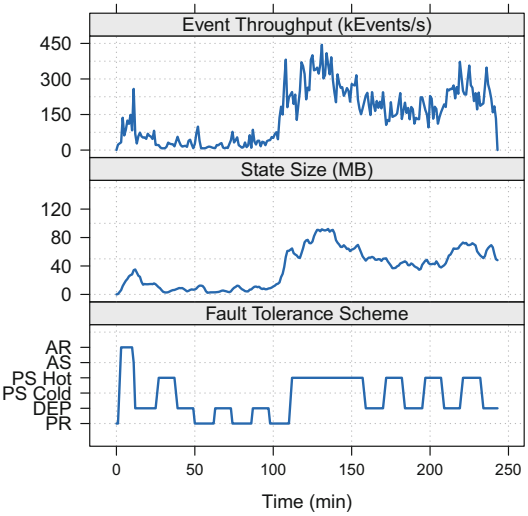
StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing, Fig. 6 Fault tolerance schemes state transition wheel: active replication, active standby, passive standby hot and cold, deployed, and passive replication

evaluated for an estimation of the expected recovery time for each of the available schemes.

Evaluation and Results

The approach has been evaluated with regard to the amount of resources that can be saved in comparison to a conservative use of full active replication. Figure 7 shows the runtime behavior of the system.

At the top graph, the throughput and how it varies over time are shown. Since the operator used for evaluation is a sliding window operator that accumulates events of the past 20 s, the size of the state follows the pattern of the throughput curve. At the bottom graphs, the chosen fault tolerance scheme is shown for each time slice. As shown in the plot, the system starts with active replication (AR), as it is the safe choice. Once enough measurements have been collected, the controller quickly switches to the deployed state replication scheme (DEP) as the state size and the throughput are quite low and, hence, recovery from disk and replay from upstream nodes can be easily accomplished within the user's specified recovery time threshold. However, as spikes occur which let the state and upstream queues grow, the controller switches between passive replication (PR) and deployed replication scheme (DEP). A cooldown time of 5 s prevents the system from oscillating due to sudden load spikes which are common in workloads originating



StreamMine3G: Elastic and Fault Tolerant Large Scale Stream Processing, Fig. 7 Throughput, state size, and fault tolerance scheme evolution over time using Twitter workload with a recovery threshold set to 5.5 s

S

from live data sources such as Twitter streams. In summary, the controller chose a combination of passive replication and deployed during the first half of the experiment, whereas the second half was dominated by passive hot standby (PS Hot).

Conclusions

This chapter presented several approaches for lowering the overhead imposed by fault tolerance mechanisms in ESP systems. First, a brief overview of StreamMine3G was provided, the ESP system used to study and evaluate the presented approaches, followed by the first approach that allowed to reduce the overhead of event ordering required to recover applications in a precise manner and to ensure reproducibility through the use of *epochs* for *commutative* and *tumbling windowed operators* (Martin et al., 2011b). In order to apply this concept also for actively replicated operators, an extension to this approach was presented where the processing of epochs is delayed and by performing an *epoch-based deterministic merge*. In conjunction with a *lightweight consensus protocol*, latency as well as the propagation of non-determinism can be reduced and prevented, respectively.

Next, an approach to increase system availability by efficiently using spare but paid cloud resources (Martin et al., 2011a) was presented. The approach combines the two replication schemes *active replication* and *passive standby* where the system transparently switches between the two states using a *priority scheduler*. The evaluation showed that the system maintains responsiveness while still providing high availability through active replication at (almost) no cost.

As a last approach, a *fault tolerance controller* (Martin et al., 2015) was presented that selects an appropriate fault tolerance scheme on behalf of the user at runtime based on previously provided constraints such as recovery time and recovery semantics. The evaluation revealed that a considerable amount of resources can be saved in comparison to the conservative use of active replication using this approach.

Cross-References

- Apache Flink
- Elasticity
- Introduction to Stream Processing Algorithms
- Apache Kafka
- Apache Samza
- Stream Benchmarks
- Types of Stream Processing Algorithms

References

Apache s4 – distributed stream computing platform (2015). https://incubator.apache.org/s4/

Apache samza – a distributed stream processing framework (2015). http://samza.incubator.apache.org/

Apache storm – distributed and faulttolerant realtime computation (2015). https://storm.incubator.apache.org/

Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

Gu Y, Zhang Z, Ye F, Yang H, Kim M, Lei H, Liu Z (2009) An empirical study of high availability in stream processing systems. In: Proceedings of the 10th ACM/IFIP/USENIX international conference on middleware, Middleware '09, New York, pp 23:1–23:9. Springer, New York

Hadoop mapreduce open source implementation (2015). http://hadoop.apache.org/

Hunt P, Konar M, Junqueira FP, Reed B (2010) Zookeeper: wait-free coordination for Internet-scale systems. In: Proceedings of the 2010 USENIX conference on USENIX annual technical conference, USENIXATC'10, Berkeley. USENIX Association, pp 1–14

Hwang J-H, Balazinska M, Rasin A, Cetintemel U, Stonebraker M, Zdonik S (2005) High-availability algorithms for distributed stream processing. In: Proceedings of the 21st international conference on data engineering, ICDE '05, Washington, DC. IEEE Computer Society, pp 779–790

Martin A, Fetzer C, Brito A (2011) Active replication at (almost) no cost. In: Proceedings of the 2011 IEEE 30th international symposium on reliable distributed systems, SRDS '11, Washington, DC. IEEE Computer Society, pp 21–30

Martin A, Knauth T, Creutz S, Becker D, Weigert S, Fetzer C, Brito A (2011) Low-overhead fault tolerance for high-throughput data processing systems. In: Proceedings of the 2011 31st international conference on dis-

tributed computing systems, ICDCS '11, Washington, DC. IEEE Computer Society, pp 689–699
Martin A, Smaneoto T, Dietze T, Brito A, Fetzer C (2015) User-constraint and self-adaptive fault tolerance for event stream processing systems. In: Proceedings of The 45th annual IEEE/IFIP international conference on dependable systems and networks (DSN 2015), Los Alamitos. IEEE Computer Society
Neumeyer L, Robbins B, Nair A, Kesari A (2010) S4: distributed stream computing platform. In: Proceedings of the 2010 IEEE international conference on data mining workshops, ICDMW '10, Washington, DC. IEEE Computer Society, pp 170–177
Schneider FB (1990) Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Comput Surv 22(4):299–319

Stream-Relational Queries

▸ Continuous Queries

StreamSQL Queries

▸ Continuous Queries

Structures for Big Data

▸ Structures for Large Data Sets

Structures for Large Data Sets

Peiquan Jin
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

Synonyms

Structures for big data; Structures for massive data

Definitions

Bloom filter (Bloom 1970): Bloom filter is a bit-vector data structure that provides a compact representation of a set of elements. It uses a group of hash functions to map each element in a data set $S = \{s_1, s_2, \ldots, s_m\}$ into a bit-vector of n bits.

LSM tree (O'Neil et al. 1996): The LSM tree is a data structure designed to provide low-cost indexing for files experiencing a high rate of inserts and deletes. It cascades data over time from smaller, higher performing (but more expensive) stores to larger less performant (and less expensive) stores.

Skip list (Black 2014): Skip list is a randomized variant of an ordered linked list with additional, parallel lists. Parallel lists at higher levels skip geometrically more items. Searching begins at the highest level, to quickly get to the right part of the list, and then uses progressively lower level lists. A new item is added by randomly selecting a level, then inserting it in order in the lists for that and all lower levels. With enough levels, the time complexity of searching is O(log$n$).

Hash table (Cormen et al. 2009): Hash table is a dictionary in which keys are mapped to array positions by hash functions. Having the keys of more than one item map to the same position is called a collision. There are many collision resolution schemes, but they may be divided into open addressing, chaining, and keeping one special overflow area.

S

Overview

This subject is mainly toward data structures for large data sets, e.g., web pages, logs, and IoT (Internet of Things) sensing data. With the rapid development of big data applications such as web and IoT applications, people have to deal with massive data that cannot be efficiently processed and stored using traditional data structures.

This entry focuses on some typical structures that have been widely used for big data representation and organization. Differing from traditional structures such as B+-tree, the structures

discussed in this entry are specially designed for large data sets. After a brief description on the key ideas of the big-data-oriented structures, some examples of application are presented. And finally, this entry suggests a few future research directions in this field.

Key Data Structures

Large data sets introduce new challenges for the underlying structures for organizing data. First, the high velocity of big data calls for write-optimized data structures that can offer a high throughput for data insertions and deletions. Second, querying processing on large data sets costs more time, which seriously throttles read performance. Thus, most of structures for large data sets aim to improve the write and read performance on big data. However, many of them focus on finding a tradeoff between read optimization and write optimization, because many structures are not possible for optimizing both read performance and write performance. On the other hand, real applications usually exhibit an asymmetric property in read and write requests, i.e., some applications are write-intensive, while others are read-intensive. To this end, we can choose specifically optimized structures for different applications on large data sets.

In this section, we present some existing structures that are proposed for read/write optimization on large data sets.

Write-Optimized B+-tree

The B+-tree is a disk-based, paginated, dynamically updateable, balanced, and tree-like index structure (Liu and Özsu 2009). It supports point queries in $O(\log_p n)$ I/Os, where n is the number of records in the tree and p is the page capacity in number of records.

While the B+-tree provides efficient and stable performance for point queries, it has poor insertion performance. Each insertion in the B+-tree has to search from the root node to the appropriate leaf node, which costs $O(\log_p n)$ I/Os. In addition, it may incur iterated update of the B+-tree in order to keep the properties of the tree such as balanced structure and approximately half filling-rate of each node.

Some write optimizations for the B+-tree are as follows (Bender and Kuszmaul 2013; Graefe 2004).

Insert in sequential order. The B+-tree is a couple of orders of magnitude faster at inserting data in sequential order compared to the random order. This is because once we insert a row into a leaf node, that leaf node is in memory, and since the next run of rows are destined for the same leaf, they can all be inserted cheaply.

This property of B+-tree leads many people to bend over backwards trying to keep insertions sequential. In many applications, this is not a natural or easy thing to do, and causes all sorts of problems elsewhere in the system.

Use a write buffer. This optimization stores up a bunch of insertions in a write buffer. When the buffer is full, we pick a leaf and write all the rows to the leaf.

This optimization works when we get to pick a bunch of stored rows that are going to the same leaf. When this happens, we see a speedup: you get to accomplish a bunch of work just for touching a leaf once.

The problem of this optimization is that we have to query the write buffer when answering queries, because the update-to-date rows may resist in the write buffer. However, as memory access is far fast than disk IOs, this approach can still have good read performance.

Write-optimized B+-trees received much attention in recent years, because of the popularity of the B+-tree in data management. It is a straightforward way to improve the B+-tree to meet the special needs of big data management. So far, write-optimized B+-trees are demonstrated advantageous for big data management, especially for streaming big data arriving at a high speed. The high throughput of insertion in write-optimized B+-trees makes it possible to meet the "velocity" challenge of big data.

The limitation of write-optimized B+-trees is that most of them sacrifice the read performance, which is not a good choice for read-intensive applications. As a result, it is necessary to devise

read/write-optimized B+-trees for big data management in future.

LSM-Tree

The Log-Structured Merge-Tree (or LSM-tree) (O'Neil et al. 1996) is a data structure proposed for highly inserted data, such as transactional log data. LSM-tree maintains data in two or more separate structures, each of which is optimized for its respective underlying storage medium. Data is synchronized in batches between the two structures efficiently.

One simple version of the LSM-tree is a two-level LSM-tree. A two-level LSM-tree comprises two tree-like structures, called C0 and C1. C0 is smaller and entirely resident in memory, while C1 is on disk. New records are first inserted into C0. If the insertion causes C0 to exceed a certain size threshold, a contiguous part of entries is removed from C0 and merged into C1 on disk. The LSM-tree has high update performance because movements from C0 to C1 are performed in batches, which implies that most writes to storage media are sequential writes.

Most implementations of LSM-tree used in practice employ multiple levels. Level 0 is kept in main memory and might be represented using a tree. The on-disk data is organized into sorted runs of data. Each run contains data sorted by the index key. A run can be represented on disk as a single file, or alternatively as a collection of files with nonoverlapping key ranges. To perform a query on a particular key to get its associated value, one must search in the Level 0 tree and each run.

The most significant advantage of the LSM-tree is its high performance of writing, because the writes from one level to its next level are always performed in sequential order. Thus, the LSM-tree is much suitable for big data applications with high-velocity data streams. The problem of the LSM-tree is its read performance. A read request in the LSM-tree may result in a few IOs to the levels which are maintained in disks.

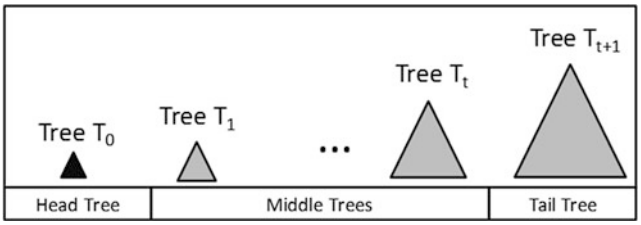
Tree Structures Optimized for New Storage

Traditional structures were mainly proposed for magnetic disks. Disk IOs causing significant access latency become a bottleneck of big data storage and management. Recently, the advent of new storage such as flash memory and phase change memory introduces new opportunities for accelerating the performance of accessing big data. In this section, some typical tree structures proposed for the context of big data and new storage are described, including FD-tree, HybridB-tree, and Bloom-tree.

FD-Tree

The idea of FD-tree is to change random writes to an index into sequential ones (Li et al. 2010). FD-tree was originally proposed for flash memory. However, as random writes are also slower than sequential writes in hard disks, FD-tree can work on different types of storage.

FD-tree is a multi-layer tree index (as shown in Fig. 1), where the top layer is a two-level B+-tree called head tree, and each of the other layers consists of a sorted list. The updates to FD-tree are initially performed on the head tree, and then are gradually moved to the sorted lists at the lower levels. The key point of FD-tree is that each sorted list is organized as sequential pages that are contiguously stored in flash memory. Therefore, when flushing the updates to the index to flash memory, FD-tree only involves sequential writes to flash memory. As sequential writes to flash memory are more efficient than random writes, FD-tree can improve the overall



Structures for Large Data Sets, Fig. 1 Structure of the FD-tree

time performance by transforming random writes to sequential ones.

HybridB Tree

The HybridB tree (Jin et al. 2015) was designed for solid-state drives (SSD) and hard disks (HDD) based hybrid storage systems. It is an optimized structure of the B+-tree. In the HybridB tree, all the interior nodes are stored on SSD. Compared with leaf nodes, interior nodes in the HybridB tree are more possible to be read, because each read to a leaf node has to incur several reads to the interior nodes on the path from the root to the leaf node. On the other hand, all the updates, insertions, and deletions to the index are first focused on leaf nodes. Although there are possible updates to some interior nodes when updating a leaf node, these updates to interior nodes are much less than those to leaf nodes. As one major objective of the HybridB tree is to reduce random writes to SSD, it is more appropriate to put leaf nodes on HDD.

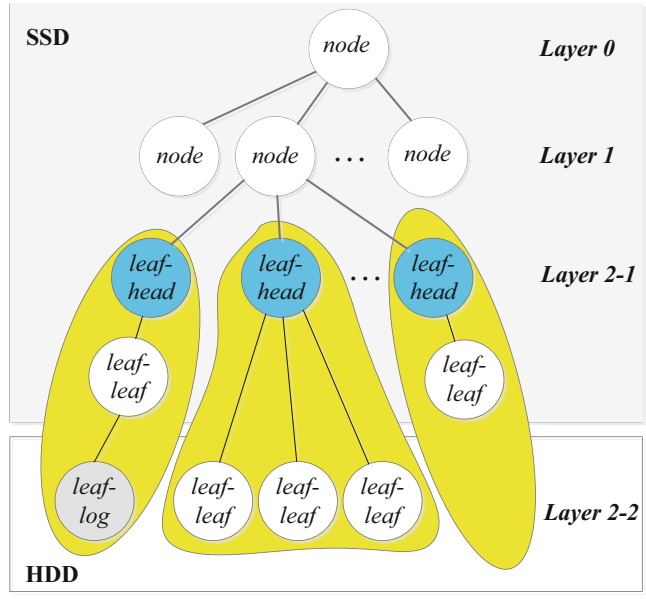
The structure of the HybridB tree is shown in Fig. 2. The leaf nodes in the tree are designed as *huge leaf* nodes (the yellow parts in Fig. 3), which are distributed among HDD and SSD. An interior node contains exactly one page, but a *huge leaf* node includes two or more pages. Each page in a *huge leaf* node can be a *leaf-head* node, a *leaf-leaf* node, or a *leaf-log* node. The reason of introducing the *huge leaf* nodes in the HybridB tree is to reduce the splits/merges to the index, because these operations will incur many random writes to SSD.

The HybridB tree has the similar search cost with the B+-tree on SSD/HDD. However, owing to the use of the huge nodes, it has fewer random writes to SSD compared with the B+-tree on SSD/HDD. Generally, the HybridB tree can achieve better overall performance than the B+-tree, especially in a big data environment, because it can get more benefits over the B+-tree when the height of the tree increases.

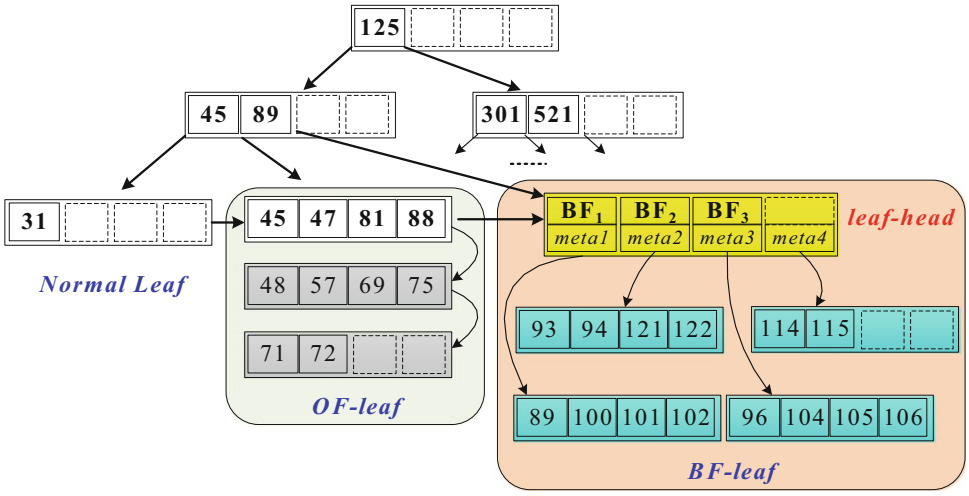
Bloom-Tree

The Bloom-tree (Jin et al. 2016) is an efficient index structure proposed for large data sets on flash memory. It is a read/write-optimized tree structure of the B+-tree.

B+-tree has been demonstrated as an efficient tree index that has fast search performance on block-based storage such as hard disks. However, updates on B+-tree will incur a substantial number of page writes to disks in order to maintain its key features of balanced tree and half node-filling. As random writes are much slow on flash



Structures for Large Data Sets, Fig. 2 Structure of the HybridB tree

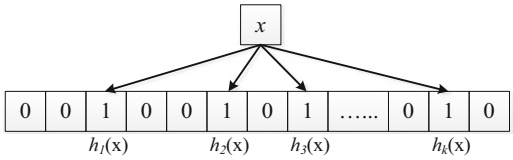


Structures for Large Data Sets, Fig. 3 Structure of the Bloom-tree

memory, many previous researches have shown that B+-tree will result in poor overall performance when directly used for flash memory (Roh et al. 2009).

In order to optimize B+-tree for flash memory, one solution is to cache the updates to the tree in a write buffer. Then, the cached updates can be merged and flushed into flash memory w.r.t. some optimal policy to reduce the overall writes on the tree index. However, even a write buffer is used for adapting B+-tree for flash memory, split operations on the tree will still lead to many page writes, as a split operation usually incurs propagating updates from a leaf node to the root of B+-tree. An intuitive way for this problem is to allow overflow pages for leaf nodes. Thus, newly inserted records can be put in overflow pages and splits will not be triggered by insertions.

So far, most flash-memory-oriented indices are designed based on the write buffer and/or the overflow page idea, aiming to reduce writes on the B+-tree. However, these solutions typically have to introduce many additional read operations. Meanwhile, modern flash disks show a close gap between their read and write speed, thus read operations on flash memory are becoming a critical factor affecting the overall time performance on flash memory. As a consequence, how to optimize flash-memory-aware indices by reducing both write and read costs is becoming an open challenge.



Structures for Large Data Sets, Fig. 4 Element map to a Bloom filter

The Bloom-tree is able to control read costs while reducing writes to flash memory. In particular, it uses an update buffer and overflow pages to reduce random writes to flash memory and further exploits bloom filter to reduce the extra reads to the overflow nodes in the tree. With this mechanism, it constructs a read/write-optimized structure that can obtain better overall performance than previous flash-aware indices.

Figure 4 shows the structure of Bloom-tree. It improves the traditional B+-tree with two new designs. First, it introduces three kinds of leaf nodes, namely *Normal Leaf*, *Overflow Leaf* (*OF-leaf*), and *Bloom-Filter Leaf* (*BF-leaf*). Second, it proposes to construct bloom filters in *BF-leaf* nodes and use overflow pages in *OF-leaf* nodes.

A normal leaf node is the same as a leaf node on the traditional B+-tree, and it occupies exactly one page. An *OF-leaf* node contains overflow

pages. However, an *OF-leaf* node contains not more than three overflow pages in order to reduce read costs on *OF-leaf* nodes. If an *OF-leaf* node expands up to more than three pages, it will be transformed into a *BF-leaf* node, which can provide more efficient organization for overflow pages. A *BF-leaf* node is designed for organizing leaves with more than three overflow pages. As shown in Fig. 3, it contains several data pages and a *leaf-head* page maintaining the bloom filters and metadata about the data.

Hashing Structures for New Storage

The idea of hashing has emerged as a basic tool in data structures and algorithms. A wide range of algorithmic problems can be solved by using hash-based structures.

A hash table is a data structure that can map keys to values (Cormen et al. 2009). A hash table uses a hash function to map a set of keys to an array of buckets. By using hash tables, people can efficiently perform searching on large data sets. For example, if you need to find out in web logs the IP address with the highest access frequency to a specific website, a hash table can be used to partition all the possible IP addresses (2^{32}) into 1024 (or more) buckets. These buckets can be maintained in memory, and the IP address with the highest frequency can be easily computed.

The hash function for a hash table is expected to map each key to only one bucket. However, in many cases, the hash function may generate the same bucket number for several keys. This causes hash collisions in hash tables (Knuth 1998). There are two basic ways to resolve hash collisions. The first way is called *separate chaining*. In this method, each bucket is appended with a list of entries. If a few keys fall in the same bucket, they will be put into the list of the bucket. The separate chaining approach will introduce extra search costs for the list lookup. The second way is called *open addressing*. In this strategy, all entry keys are stored in the bucket itself. When a new key has to be inserted, the buckets are examined, starting with the hashed-to bucket and proceeding in some probe sequence, until an unoccupied bucket is found. When searching for an entry, the buckets are scanned in the same sequence, until either the target record is found or an unused bucket is found, which indicates that there is no such key in the hash table. The name "*open addressing*" means that the address of a key is not determined by its hash value.

In recent years, there are some studies focusing on optimizing hashing structures for new storage such as flash memory. Flash memory has faster read speed than disks, but its random write speed is slower. Thus, hashing structures for flash memory mostly aim to reduce random writes to flash memory. MicroHash (Zeinalipour-Yazti et al. 2005) is an efficient external memory index structure for wireless sensor devices, which stores data on flash by time or value. MLDH (Multi Level Dynamic Hash index) (Yang et al. 2009) is a multi-layered hash index, and the capacity of each level in MLDH is twice as its upper level. Updates to MLDH are first buffered in an in-memory structure. When the memory structure is full, the memory data are merged with the hash index on flash memory and a new hash index is built. Wang et al. proposed a flash-based self-adaptive extendible hash index where each bucket occupies a block (erase unit) of flash memory (Wang and Wang 2010). A bucket consists of both data region and log region. The log region serves updates and deletes, so in-place updates to the data region can be delayed. In addition, a Split-or-Merge (SM) factor, which is dynamically adjusted according to the log/data ratio, is introduced to make the index self-adaptive. Hybrid Hash Index (Yoo et al. 2012) delays split operations which cause additional writes and erasure operations by using overflow buckets. Li et al. proposed a lazy-split hashing scheme to reduce writes at the expense of more reads (Li et al. 2008). They declared that the lazy-split hashing can adapt itself dynamically to different search ratios. Yang et al. proposed SAL-Hashing (Self-Adaptive Linear Hashing) (Yang et al. 2016) to reduce small random-writes to flash memory that are caused by index operations.

In-Memory Structures for Large Data Sets

In-memory data placement and processing has been regarded as a basic principle for ensuring fast read/write speed in big data management.

For this purpose, efficient in-memory structures are needed to be specially designed. This issue is especially important as the memory size is increasing with time. In this section, several structures proposed for in-memory data placement and accesses are discussed, including Bloom filter and Skip list.

Bloom Filter

Bloom filter is a space-efficient in-memory data structure to represent a set to which we can add elements and answer membership queries approximately (Bloom 1970). It has been extensively used as auxiliary data structures in database systems. Bloom filter is a bit-vector data structure that provides a compact representation of a set of elements. It maps each element in a data set $S = \{s_1, s_2, \ldots, s_m\}$ into a bit-vector of n bits, which is denoted by BF[1], ..., BF[n] with a default value of 0. In this mapping procedure, we use a group of independent hash functions $\{h_1, h_2, \ldots, h_k\}$. For a given element in S, each hash function maps it into a random number within the set of $\{1, 2, \ldots, n\}$. If a hash function returns i, we set BF[i] to 1. Figure 4 shows the mapping idea of Bloom filter.

For an element $x \in S$, the algorithm to compute its bloom filter includes two steps. First, it calculates the k values of hash functions $h_1(x)$, $h_2(x)$, $\ldots$, $h_k(x)$. Second, it sets all bits BF[$h_i(x)$] to 1.

For answering a membership query like "Is $y \in S$?," the algorithm first calculates the k values of hash functions $h_1(y)$, $h_2(y)$, $\ldots$, $h_k(y)$. Then, it checks all the Bloom filters of each element in S to see whether all the BF[$h_i(y)$] in an existing Bloom filter are 1. If not, y is not a member of S. If all the BF[h_i(y)] are 1, y may be within S. Due to the possible collisions of hash functions, there is a *false positive* rate when evaluating membership queries on Bloom filters. Given n, k, and m, the false positive probability of a Bloom filter can be computed by Eq. (1). Further, it is demonstrated that the false positive probability is minimalized when $k = 0.7 \cdot \frac{m}{n}$, which is approximately $0.6185^{\frac{m}{n}}$ (Bloom 1970).

$$f^{BF} = \left(1 - e^{-\frac{nk}{m}}\right)^k \tag{1}$$

Bloom filter is space efficient in representing elements. In addition, it is time efficient for inserting elements and answering membership queries. However, Bloom filter does not support deletions on elements. A revised version enhances Bloom filter with deletions (Bonomi et al. 2006).

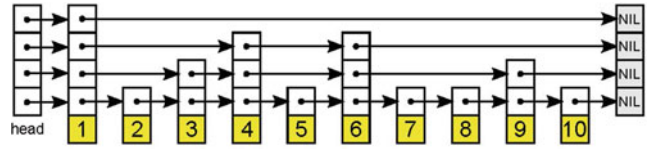
Skip List

Skip list is a linked-list-like in-memory structure that allows fast search in memory (Pugh 1990). It consists of a base list holding the elements, together with a tower of lists maintaining a linked hierarchy of subsequences, each skipping over fewer elements.

Skip list is a data structure that can be used in place of balanced trees. It uses probabilistic balancing rather than strictly enforced balancing and as a result the algorithms for insertion and deletion in skip lists are much simpler and significantly faster than equivalent algorithms for balanced trees. Skip lists are built in layers, as shown in Fig. 5. The bottom layer is an ordinary ordered linked list. At a high level, a skip list is just a sorted, singly linked list with some "shortcuts" (additional pointers that allow traveling faster. An element in layer i appears in layer $i + 1$ with some fixed probability p (two commonly used values for p are1/2 or 1/4). On average, each element appears in 1/(1-p) lists, and the tallest element (usually a special head element at the front of the skip list) in all the lists.

A search for a target element begins at the head element in the top list and proceeds horizontally until the current element is greater than or equal to the target. If the current element is equal to the target, it has been found. If the current element is greater than the target, or the search reaches the end of the linked list, the procedure is repeated after returning to the previous element and dropping down vertically to the next lower list.

Indeed, a skip list is really just a simulation of a binary search tree using multiple linked lists running in parallel. Of course, trying to maintain a perfectly balanced binary search tree using this simulation is still expensive, and it is still complicated. However, by viewing a skip list as



Structures for Large Data Sets, Fig. 5
Structure of skip lists

a single sorted linked list, where each node has a column of links to other nodes in the list, it is easy to see how randomly choosing a height for each new column could potentially produce the same good behavior as a balanced skip list.

The original design of the skip list was as a randomized data structure, much like randomized binary search trees. By eschewing a logarithmic bound guarantee in favor of a logarithmic bound with high probability, a skip list can enjoy the performance properties of a well-balanced binary search tree (on average) while avoiding many disadvantages of tree structures.

Compared with the balanced trees, the skip list has the following advantages:

1. It is relatively easy to design and implement the algorithm. The implementation of the skip list is direct and easier than the balanced trees.
2. It is efficient. Insertion and deletion do not need to be balanced again. Indeed, the skip list serves as an alternative to the binary search trees which become unbalanced after several insertion and deletion operations.
3. The memory requirements of skip lists are less than that used for balanced trees.

In the big data scope, the skip list is commonly employed as an efficient structure for organizing memory data. For example, it has been implemented in a couple of NoSQL database engines like LevelDB, Redis, and MemSQL as the primary memory structure.

Examples of Application

In many cases, hash tables are more efficient than trees or any other table structures. For this reason, they are widely used in many kinds of computer software, particularly for associative arrays, database indexing, caches, and large sets. Recent research by Google (Henzinger 2006; Das et al. 2007) showed that using deep learning approach can optimize hash functions by orders of magnitude in space usage savings and improve retrieval speed as well.

Hash tables may also be used as disk-based data structures and database indices. In multi-node database systems, hash tables are commonly used to distribute rows among nodes, reducing network traffic for hash joins (Karger et al. 1997).

Hashing is also commonly used in information retrieval. One early application in this field is identifying near-duplicate web pages in Altavista using min-wise hashing (Broder et al. 1998). Another application is HyperANF (Boldi et al. 2011), which was used to compute the distance distribution of the Facebook social network. In the field of machine learning, random mappings of data to lower-dimensional vectors that are easier to handle is of increasing importance for big data applications. This is because that machine learning algorithms often work with kernelized feature vectors with high dimensions. Designing randomized mappings that meet the criteria of machine learning applications has been an active research area in recent years (Wang et al. 2016).

A key application of Bloom filters is in the field of content delivery networks, which deploys web caches around the world to cache and serve web content to users with greater performance and reliability. Bloom filters are used to efficiently determine which web objects to store in web caches (Maggs and Sitaraman 2015). To prevent caching one-hit-wonders (accessed by users only once and never again), a Bloom filter is used to keep track of all URLs that are accessed by users. A web object is cached only when it has been accessed at least once before. With this mechanism, one can significantly reduce the disk write workload, since one-hit-wonders are never written to the disk cache. Further, filtering out the

one-hit-wonders also saves cache space on disk, increasing the cache hit rates.

Bloom filters can also be organized as distributed data structures to perform fully decentralized computations of aggregate functions (Pournaras et al. 2013). Decentralized aggregation makes collective measurements locally available in every node of a distributed network without involving a centralized computational entity for this purpose.

The LSM-tree is now used in many database products such as Bigtable, HBase, LevelDB, MongoDB, SQLite, RocksDB, WiredTiger, Apache Cassandra, and InfluxDB. The LSM-tree is especially suitable for write-intensive workloads. Certainly, LSM implementations such as LevelDB and Cassandra regularly provide better write performance than single-tree based approaches. Yahoo! developed a system called PNUTS, which combines the LSM-tree with the B-trees and demonstrates better performance (Cooper et al. 2008).

It is also worth considering the hardware being used. Some expensive solid state disks, like FusionIO, have better random write performance. This suits update-in-place approaches. Cheaper SSDs are better suited to the LSM-tree, because it can avoid small random writes to SSDs.

Skip list has been implemented in many systems such as LevelDB, MemSQL, and Redis. Specially, Redis adds a backward pointer for each skip list node to traverse reversely. Also there is a span variable in level entry to record how many nodes must be crossed when reaching to next node. Actually, when traverse list, we can accumulate span to get the rank of a node in sorted set. In LevelDB, skip lists are used to organize the MemTable in memory. In MemSQL, skip lists are used as the prime indexing structure for databases.

Future Directions for Research

Data structures are one of the most critical parts for big data representation, processing, and storage. In addition to the structures explained in this article, there are some future research directions in this field.

Firstly, the existing tree structures like LSM-tree and HybridB-Tree are mainly toward write-intensive workloads. Although it is a major objective to reduce random writes to new storage such as flash memory, it is valuable to devise read/write-friendly structures in the future.

Secondly, traditional database researches suppose that data are stored in external disks. However, big data applications call for in-memory data processing and storage. Thus, how to make data structures suit for in-memory data management is an important issue in the big data era.

Finally, recently nonvolatile memory (NVM) has received much attention from both academia and industries (Li et al. 2016; Chen et al. 2014). This will lead to reformation of memory architecture. Therefore, developing NVM-oriented structures is a potential research direction in the future.

Acknowledgments

We would like to thank University of Science and Technology of China for providing the environment where the study described in this entry was completed. The work involved in this entry is partially supported by the National Science Foundation of China (No. 61672479).

Cross-References

- Big Data Indexing
- Search and Query Accelerators

References

Bender M, Kuszmaul B (2013) Data structures and algorithms for big databases. In: 7th extremely large databases conference, Workshop, and Tutorials (XLDB), Stanford University, California

Black P (2014) Skip list. In: Pieterse V, Black P (eds) Dictionary of algorithms and data structures. https://www.nist.gov/dads/HTML/skiplist.html

Bloom B (1970) Space/time trade-offs in hash coding with allowable errors. Commun ACM 13(7):422–426

Boldi P, Rosa M, Vigna S (2011) HyperANF: approximating the neighbourhood function of very large graphs on a budget. In: Srinivasan S et al (eds) Proceedings of the 20th international conference on World Wide Web, March 2011, Hyderabad/India, p 625–634

Bonomi F, Mitzenmacher M, Panigrahy R, Singh S, Varghese G (2006) An improved construction for counting Bloom filters. In: Azar Y, Erlebach T (eds) Algorithms – ESA 2006, the 14th annual european symposium on algorithms, September 2006, LNCS 4168, Zurich, Switzerland, p 684–695

Broder A, Charikar M, Frieze A, Mitzenmacher M (1998) Min-wise independent permutations. In: Vitter J (eds) Proceedings of the thirtieth annual ACM symposium on the theory of computing, May 1998, Dallas, Texas, p 327–336

Chen K, Jin P, Yue L (2014) A novel page replacement algorithm for the hybrid memory architecture involving PCM and DRAM. In: Hsu C et al (eds) Proceedings of the 11th IFIP WG 10.3 international conference on network and parallel computing, September 2014, Ilan, Taiwan, p 108–119

Cooper B, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen H, Puz N, Weaver D, Yerneni R (2008) PNUTS: Yahoo!'s hosted data serving platform. Proc VLDB Endowment 1(2):1277–1288

Cormen T, Leiserson C, Rivest R, Stein C (2009) Introduction to algorithms, 3rd edn. MIT Press, Boston, pp 253–280

Das A, Datar M, Garg A, Rajaram S (2007) Google news personalization: scalable online collaborative filtering. In: Williamson C et al (eds) Proceedings of the 16th international conference on World Wide Web, May 2007, Banff, Alberta, p 271–280

Graefe G (2004) Write-Optimized B-Trees. In: Nascimento M, Özsu M, Kossmann D, et al. (eds) Proceedings of the thirtieth international conference on very large data bases, Toronto, Canada, p 672–683

Henzinger M (2006) Finding near-duplicate web pages: a large-scale evaluation of algorithms, In: Efthimiadis E et al (eds) Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval, August 2006, Seattle, Washington, p 284–291

Jin P, Yang P, Yue L (2015) Optimizing B+-tree for hybrid storage systems. Distrib Parallel Databases 33(3): 449–475

Jin P, Yang C, Jensen C, Yang P, Yue L (2016) Read/write-optimized tree indexing for solid-state drives. VLDB J 25(5):695–717

Karger D, Lehman E, Leighton T, Panigrahy R, Levine M, Lewin D (1997) Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: Leighton F et al (eds) Proceedings of the twenty-ninth Annual ACM symposium on the theory of computing, May 1997, El Paso, Texas, p 654–663

Knuth D (1998) The art of computer programming. 3: sorting and searching, 2nd edn. Addison-Wesley, New York, pp 513–558

Li X, Da Z, Meng X (2008) A new dynamic hash index for flash-based storage. In Jia Y et al (eds) Proceedings of the ninth international conference on web-age information management, July 2008, Zhangjiajie, China, p 93–98

Li Y, He B, Yang J, Luo Q, Yi K (2010) Tree indexing on solid state drives. Proc VLDB Endowment 3(1): 1195–1206

Li L, Jin P, Yang C, Wan S, Yue L (2016) XB+-tree: a novel index for PCM/DRAM-based hybrid memory. In: Cheema M et al (eds) Databases theory and applications – proceedings of the 27th Australasian database conference, September 2016, LNCS 9877, Sydney, Australia, p 357–368

Liu L, Özsu M (2009) Encyclopedia of database systems. Springer, New York

Maggs B, Sitaraman R (2015) Algorithmic nuggets in content delivery. SIGCOMM Comput Commun Rev 45(3):52–66

O'Neil P, Cheng E, Gawlick D, O'Neil E (1996) The log-structured merge-tree (LSM-tree). Acta Informatica 33(4):351–385

Pournaras E, Warnier M, Brazier F (2013) A generic and adaptive aggregation service for large-scale decentralized networks. Complex Adapt Syst Model 1:19

Pugh W (1990) Skip lists: a probabilistic alternative to balanced trees. Commun ACM 33(6):668

Roh H, Kim W, Kim S, Park S (2009) A B-tree index extension to enhance response time and the life cycle of flash memory. Inf Sci 179(18): 3136–3161

Wang L, Wang H (2010) A new self-adaptive extendible hash index for flash-based DBMS. In Hao Y et al (eds) Proceedings of the 2010 IEEE international conference on information and automation, June 2010, Haerbin, China, p 2519–2524

Wang J, Liu W, Kumar S, Chang S (2016) Learning to hash for indexing big data – a survey. Proc IEEE 104(1):34–57

Yang C, Lee K, Kim M, Lee Y (2009) An efficient dynamic hash index structure for NAND flash memory. IEICE Trans Fundam Electron Commun Comput Sci 92(7):1716–1719

Yang C, Jin P, Yue L, Zhang D (2016) Self-adaptive linear hashing for solid state drives. In Hsu M et al (eds) Proceedings of the 32nd IEEE international conference on data engineering, May 2016, Helsinki, Finland, p 433–444

Yoo M, Kim B, Lee D (2012). Hybrid hash index for NAND flash memory-based storage systems. In: Lee S et al (eds) Proceedings of the 6th international conference on ubiquitous information management and communication, February 2012, Kuala Lumpur, Malaysia, p 55:1–55:5

Zeinalipour-Yazti D, Lin S, Kalogeraki V, Gunopulos D, Najjar W (2005) MicroHash: an efficient index structure for flash-based sensor devices. In: Gibson G (eds) Proceedings of the FAST '05 conference on file and storage technologies, December 2005, San Francisco, California, p 1–14

Structures for Massive Data

▸ Structures for Large Data Sets

Survey

▸ Big Data in Computer Network Monitoring

SUT

▸ System Under Test

SWAG

▸ Sliding-Window Aggregation Algorithms

System Under Test

Francois Raab
InfoSizing, Manitou Springs, CO, USA

Synonyms

Measured system; SUT; Target environment; Test object; Tested environment

Definitions

A System Under Test (SUT) is a complete system that comprises hardware, software, and connectivity components; that is the object or target of a performance measurement test or benchmark.

Historical Background

While the term System Under Test is somewhat generic in nature, it was formalized in the early 1990s by industry consortiums with the mission of defining industry standard performance benchmarks, such as the Transaction Processing Performance Council (TPC – www.tpc.org), the Standard Performance Evaluation Corporation (SPEC – www.spec.org), and the Storage Performance Council (SPC – www.storageperformance.org).

Foundations

The purpose of a test or benchmark is to determine how a target environment behaves under a controlled load or in response to specific requests. As such, the definition of the System Under Test plays a key role in a benchmark specification. The SUT's definition discusses the inclusion or exclusion of hardware, software, and connectivity components. It also defines the measurement boundaries. For benchmarks that mandate some pricing metrics, the SUT defines which components must be included in this pricing.

Hardware Components

The definition of the SUT includes a number of hardware components. These components can be specific or generic. Specific components can be chosen to define a fixed set of hardware on which to measure different software variations. By keeping the hardware as a constant while varying the software, the outcome of the test can be fully attributed to changes in the software environment. Alternatively, the hardware components can be defined in generic terms, giving some latitude in their selection. This latitude can be limited to some portion of the SUT or can be applied to all hardware components in the SUT. For example, the SUT's definition may require the configuration of a single server node with specific attached storage and a specific number of processor sockets, while giving latitude in the choice of processor types. Such a SUT definition would focus the results of the test of the

processor's capabilities. Alternatively, the SUT's definition may give full latitude in the selection of the server's internal architecture, system topology (single-node, cluster, client–server, etc.), and storage subsystem; giving the test an opportunity to compare widely diverging solutions to a common workload requirement, or analyzing non-hardware criteria, such as price, value (price/performance), scalability, or portability.

Software Components

The definition of the SUT generally includes a number of software components or a software stack. Few tests solely focused on a hardware component (e.g., a simple storage device) and do not require any software as part of the SUT. But, in most cases, one or more software components are needed to assemble the SUT. There is a degree to the role that software can play in the SUT. Software can be part of the SUT to simply tie together the hardware components targeted by the test. In the case of system testing, software components are just as important as hardware components for the tested system, literally a System Under Test. And for tests that directly target software, rather than hardware, the software components are the object of the test and the hardware components merely provide a platform for the software to execute.

Connectivity Components

The definition of the SUT may include a number of connectivity components. These are used to allow the hardware components to communicate. Such connectivity might be in the form of a general-purpose network (e.g., Ethernet over Fibre Channel) or in the form of a proprietary interconnect (e.g., AMD's HyperTransport bus). Many test environments use some connectivity components to provide a communication channel between the SUT and the test harness. These communication channels may or may not be included as part of the SUT. One criteria to include such communication channel in the SUT is whether they are also used for communication between other components inside the SUT.

Measurement Boundary

The boundary between a test driver (also called measurement harness) and the target of the measurement (i.e., the SUT) is the point where the driver applies a stimulus to the SUT and where the driver measures the SUT's response to that stimulus. This measurement boundary is also where the test driver ends and the SUT starts. In some cases, the measurement boundary is virtual in that the driver uses resources from components that are part of the SUT. In other cases, the boundary is physical in that no components, other than a communication channel, are shared between the driver and the SUT.

Pricing Metrics

When benchmarks results are used for competitive analysis, it may be relevant to include some pricing metrics. The purpose of such metrics is to provide information about the cost associated with the performance capabilities of the SUT. These metrics are generally expressed in terms of straight total price, in terms of ratio between total price and measured performance, or both. For example, the TPC-A metrics include the total cost of ownership of the SUT and the associated cost per unit of performance, expressed in cost/tps (transaction per second). The definition of the SUT is a key to defining which components are included in these cost calculations.

Key Applications

First Formal Use

The first formal use of the term SUT was found as part of the initial release of TPC Benchmark A (TPC-A) (Huppler 2009), published by the TPC in 1990. This online transaction processing (OLTP) benchmark provided an industry-sanctioned specification of the loosely defined and widely used DebitCredit (Anon 1985) workload. The TPC-A specification defined the term SUT as including the following components:

- *One or more processing units (e.g., hosts, front-ends, and workstations.), which will run the transactions [. . .].*

- *The hardware and software components of all networks required to connect and support the SUT components.*
- *Data storage media sufficient to satisfy both the scaling rules [. . .] and the ACID properties [. . .].*
- *The host system(s) including hardware and software supporting the database employed in the benchmark.*

Generic Terminology

Following in the footsteps of TPC-A, subsequent benchmark specifications released by industry standard consortia have been using the term SUT, or one of its synonyms, to identify the performance measurement target specific to each benchmark. The term SUT, by itself, does not describe the tested environment. Instead, the term is used as a container to encapsulate the set of hardware, software, and connectivity components that are being targeted for measurement. As such, each benchmark specification includes its own formal definition of the term SUT.

In some instances, the term used to name the target environment of a test is synonymous with System Under Test. For example, SPEC does not consistently use the term SUT in its benchmark specifications. However, the consortium uses the term SUT to name the set of configuration components that can be chosen as filters for queries against its database of benchmark publications.

Benchmark Adoption and Longevity

The SUT's definition is a key factor in a benchmark's adoption and longevity. According to (Huppler 2009), all "good" benchmarks share the same primary characteristics of being *relevant, repeatable*, *fair*, *verifiable*, and *economical*. A careful definition of the SUT will contribute to improving each one of these characteristics.

- Relevant: The SUT's primary components are those for which test results are in demand.
- Repeatable: The SUT's definition has tight requirements for components that impact the test results, and gives flexibility for components that are result neutral.
- Fair: Components of the SUT are limited to those that can be compared on an even playing field.
- Verifiable: The availability of the components used and their configuration is such that others can easily recreate the SUT to repeat the test.
- Economical: The cost of assembling the SUT does not create a prohibitive limit on who can reasonably execute the test.

Cross-References

- ▸ Big Data Benchmark
- ▸ Benchmark Harness
- ▸ Big Data Architectures
- ▸ Cloud Big Data Benchmarks
- ▸ Component Benchmark
- ▸ End-to-End Benchmark
- ▸ Energy Benchmarking
- ▸ Metrics for Big Data Benchmarks
- ▸ Microbenchmark
- ▸ TPC

References

Anon et al (1985) A measure of transaction processing power. Datamation 31:112–118

Huppler K (2009) The art of building a good benchmark. In: Nambiar R, Poess M (eds) Performance evaluation and benchmarking, vol 5895. Springer, Berlin/Heidelberg, pp 18–30. https://link.springer.com/chapter/10.1007/978-3-642-10424-4_3