A Survey on Big Data Analytics Technologies

Fei Su[1(✉)], Zhenya Wang[1], Shan Yang[1], Ke Li[1], Xin Lu[1], Yang Wu[1], and Yi Peng[2]

[1] China Unicom Network Technology Research Institute, Beijing, China
sufei@chinaunicom.cn
[2] National Satellite Meteorological Center, Beijing, China

Abstract. With the beginning of new era, data has grown rapidly in both the size and the variety. It becomes not only an important cornerstone of all walks of life, but also the national strategy. The big data collection, parsing, analysis, and applications are important issues to research. For different scenarios of big data applications, appropriate big data processing technologies are needed to complete the real-time and rapid data analysis. The objective of this paper is to analyze the typical big data analysis technologies, find out the characteristics and applicative scenarios, and then provide the reference for big data processing of all industries.

Keywords: Big data · Hadoop · Spark · NoSQL · Streaming · MPP database

1 Introduction

With the development of big data, a variety of data analysis technologies are arisen. These technologies are different from the traditional methods of data statistics and data mining. They have different characteristics and applicable scenarios. To research the characteristics of big data technologies is a key step to the deployment of big data strategy and data realization.

Traditional data warehouse architecture based on Oracle is no longer adapted to the current needs of big data analysis [1]. The open source data analysis architecture is the main stream. Currently, there are variety of big data analysis techniques [2]. Hadoop is an open-source software framework for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework. Spark is a fast and general engine for large-scale data processing. Its performance is better than Hadoop, because the calculation is processing in memory. Storm is a free and open source distributed real-time computation system. Storm makes it easy to reliably process unbounded streams of data, doing for real-time processing what Hadoop did for batch processing. Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Twitter, ZeroMQ, Kinesis, or TCP sockets. Impala (incubating) is the open source, native analytic database for Apache Hadoop. HBase is an open-source, distributed, versioned, non-relational database

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2018
K. Long et al. (Eds.): 5GWN 2017, LNICST 211, pp. 359–370, 2018.
https://doi.org/10.1007/978-3-319-72823-0_34

modeled after Google's Bigtable: A Distributed Storage System for Structured Data by Chang et al. Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

How to effectively combine these big data techniques, and achieve the data's parsing, storage, and analysis for different industries is an important issue to research. The mainstream big data architecture usually adopts mixed style [3, 4]. Firstly, the HDFS [5, 6] is used to carry out the underlying storage. Secondly, the MR/SPARK is used to accomplish batch-processing. Finally, the resulting data set is stored in the traditional relational database, such as oracle. In this paper, we analyze the typical big data analysis technologies, find out the characteristics and applicative scenarios, and then provide the reference for big data processing of all industries.

2 Hadoop

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. As a result, we can easily write distributed programs which fully take advantage of the cluster power to do the computing and storage work without knowing the details of the lower-level of the distributed system. Compared with the other systems, Hadoop has several advantages [7, 8] such as higher reliability, higher extendibility, higher efficiency, higher fault-tolerance and lower cost, etc. (Fig. 1).

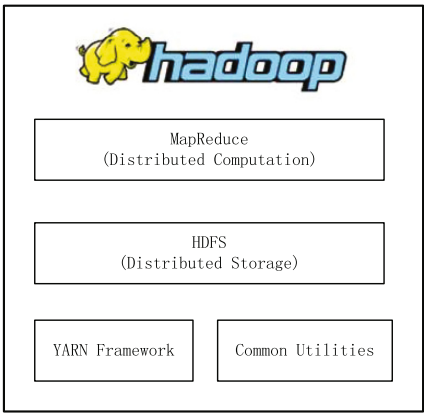

Fig. 1. Hadoop base framework

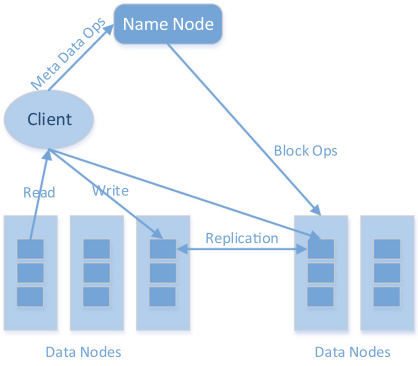

Fig. 2. HDFS architecture [9]

2.1 Hadoop Architecture

Hadoop framework mainly contains four modules:

- **MapReduce** is a YARN-based framework for parallel processing of large data sets in a reliable manner.

- **HDFS (Hadoop Distributed File System)** is a distributed file system that provides high-throughput access to application data.
- **Hadoop Common** contains Java libraries and utilities required by other Hadoop modules. These libraries include OS level abstractions and necessary files and scripts to start Hadoop.
- **YARN** is responsible for job scheduling and cluster resource management.

These four base modules provide base function of Hadoop system. We will focus on the HDFS and MapReduce in this section.

2.2 HDFS

As one of the top-level project in the Hadoop eco-system, HDFS is the fundamental of the distributed data storage system. It is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a master/slave architecture which is shown in Fig. 2. The master machine runs a NameNode software and is responsible for the management of the file system namespace and the regulation of the file access. The slave machines run a DataNodes software and perform read-write operation on the file systems according to the client requests, do block creation, deletion, and replication under the instructions of the NameNode. We should notice that there is no practical difference between master and slaver machines, they just run different software to perform different role in the HDFS system.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode (Fig. 3).

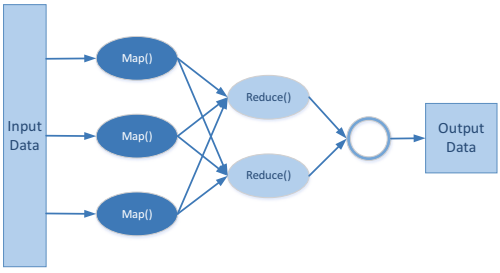

Fig. 3. Map and Reduce stage

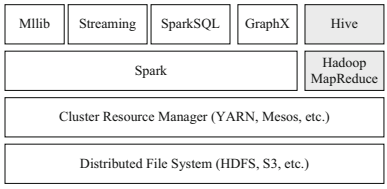

Fig. 4. Typical Spark deployment plan

2.3 MapReduce

In 2004, Google proposed MapReduce as a parallel computation model in order to solve the calculation of the big data analysis problem. MapReduce module separates

the parallel computation process into two stage: Map task and Reduce task [10]. Firstly, the set of data is collected and converts it into another set of data by map task. These data elements are broken down into key and value pairs. Secondly, the task reduction takes the output from a map as an input and combines those data tuples into a smaller set of tuples. Typically both the input and the output are stored in a file-system. The MapReduce framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

By adopting MapReduce technique, it is much eases to scale data processing over multiple computing nodes: Decomposing a data processing application into mappers and reducers is normally a time-consuming work. However, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

3 Spark

3.1 Spark Framework

Spark is an in-memory parallel computing framework which specialized in fast data analyzing on large-scale dataset. Thanks to its highly active open-source community, Spark contains several original libraries supporting structured data processing (SparkSQL), machine learning (MLlib), streaming (Spark Streaming) and graph-parallel computation (GraphX). It also supports various languages including Scala, Java, Python and R. Spark was first developed in UC Berkeley in 2009 and is currently hosted by Apache Software Foundation with over 1000 contributors worldwide.

As a data processing framework, Spark extends the MapReduce model to handle more complicated tasks, especially iterative computing and interactive analytics [11]. The former is commonly found in machine learning algorithms, and the latter is used in SQL queries on large datasets. On the other hand, Spark inherits those good features from the MapReduce model such as scalability and fault tolerance. As shown in Fig. 4, Spark is compatible with some core components of the Hadoop ecosystem including HDFS, Hive and YARN, making it one of the most universal data processing systems.

3.2 Spark Principles

The key abstraction used in Spark is called Resilient Distributed Dataset (RDD), which is a collection of partitioned data stored in memory across the nodes in the cluster and is able to be kept in memory for future reuse. A lineage mechanism is used to keep track of the data transformations to recovery from failure. In the typical cluster mode, a Spark application runs as the following procedures:

1. A SparkContext object is first created in the driver program, which acts as the controller of the entire application.

2. Next, the SparkContext contacts with the cluster resource manager, which assigns all the cores, memory and network IO required by the application.
3. Once connected, Spark acquires Executors on each worker node, which are collections of computation resources and the application code itself. Each application has a set of its own Executors.
4. Finally, the SparkContext sends different Tasks to each Executor to run the actual computation (Fig. 5).

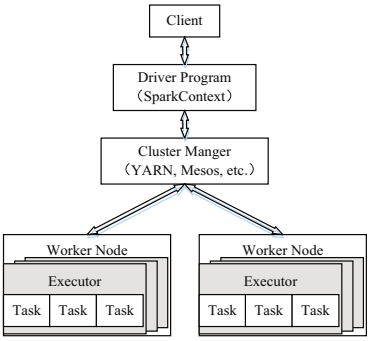

Fig. 5. Spark application running procedures

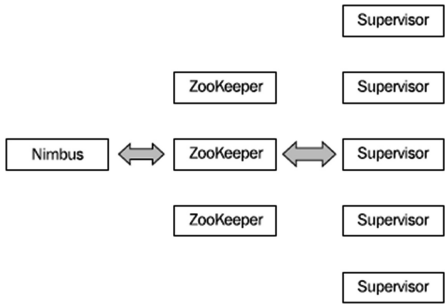

Fig. 6. Storm cluster architecture

The vanilla Spark supports more than 80 basic operators on RDDs, including map, filter, union, join, groupByKey, reduceByKey, etc. On top of RDD, a higher level data structure, DataFrame, is used in SparkSQL (and other modules as well) to deal with structured data processing. A DataFrame is similar to a table in a relational database, and can be created from either HDFS files or Hive tables. One can perform SQL queries directly on DataFrames, making it much easier analyzing data stored in an existing data platform. Likewise, Spark Streaming and MLlib also implement corresponding high-level APIs for real-time and machine learning applications.

3.3 Performance

According to an early research, native Spark runs 25× faster than Hadoop MapReduce on a Logistic Regression application. In another benchmark dealing with SQL queries, SparkSQL is generally competitive with Impala, an in-memory MPP database built on Hadoop [12]. Such result is remarkable considering Impala is written in C++ and is well-optimized for SQL queries, whereas Spark is a general data processing engine and runs in JVM. As for graph processing, Spark GraphX achieve a similar runtime performance compared with GraphLab and Giraph, and slightly better scaling performance than the latter two systems.

In general, Spark is a multi-functional data analyzing framework that excels in speed, scalability and reliability. It is compatible with Hadoop system and runs much faster than the MapReduce-based programs. Plus, Spark incorporates a series advanced APIs which significantly enhance the performance of big data applications such as streaming, SQL queries, machine learning and graph analysis.

4 Stream

Streaming data as a new form of large data more reflects the characteristics of high-speed and real-time processing. There are two ways to calculate large data: batch computing and stream computing. Streaming data is disordered and ineffective, so you can't store all data as in batch computing model. In the stream computing, streaming data is stored in memory for real-time computing. Existing streaming computing platforms are Storm, Spark streaming, Samza and so on, which are open-source distributed systems, with low latency, scalability and fault tolerance and many other advantages, allowing you to run the data stream code, tasks will be assigned to a series of fault-tolerant nodes on the parallel operation.

4.1 Storm

Apache Storm is a free and open source distributed real-time computation system for processing streams of data. Storm makes it easy to reliably process unbounded streams of data, doing for real-time processing what Hadoop did for batch processing. A stream is an unbounded sequence of tuples that is processed and created in parallel in a storm cluster. Every stream is given an id when declared. The work to process stream is delegated to different types of components that are each responsible for a simple specific processing task.

The input stream of a Storm cluster is handled by a component called a spout [13]. A spout is a source of streams in a topology. Generally spouts will read tuples from an external source and emit them into the topology. The spout passes the data to a component called a bolt, which transforms it in some way. Storm guarantees that every spout tuple will be fully processed by the topology. It does this by tracking the tree of tuples triggered by every spout tuple and determining when that tree of tuples has been successfully completed. Every topology has a "message timeout" associated with it. If Storm fails to detect that a spout tuple has been completed within that timeout, then it fails the tuple and replays it later.

A bolt either persists the data in some sort of storage, or passes it to some other bolt, which can do anything from filtering, functions, aggregations, joins, talking to databases, and more. Bolts can do simple stream transformations, while doing complex stream transformations often requires multiple steps and thus multiple bolts.

The logic for a real-time application is packaged into a Storm topology. A Storm topology is analogous to a MapReduce job. One key difference is that a MapReduce job eventually finishes, whereas a topology runs forever or until it is killed. A topology is a graph of spouts and bolts that are connected with stream groupings.

As Fig. 6 shows, in a Storm cluster [14], nodes are organized into a master node that runs continuously. There are two kind of nodes in a Storm cluster: master node and worker nodes. Master node run a daemon called Nimbus, which is responsible for distributing code around the cluster, assigning tasks to each worker node, and monitoring for failures. Worker nodes run a daemon called Supervisor, which executes a portion of a topology. A topology in Storm runs across many worker nodes on different machines. Since Storm keeps all cluster states either in Zookeeper or on local disk, the daemons are stateless and can fail or restart without affecting the health of the system.

The following are the features of storm [15]:

- Fast: Storm can process up to 1 million tuples per second per node.
- Horizontally scalable: being fast is a necessary feature to build a high volume/velocity data processing platform, but a single-node will have an upper limit on the number of events that it can process per second. Storm is linearly scalable. More nodes can be added to a Storm cluster to increase the processing capacity of applications.
- Fault tolerant: units of work are executed by worker processes in a Storm cluster. When a worker dies, Storm will restart that worker, and if the node on which the worker is running dies, Storm will restart that worker on some other node in the cluster.
- Guaranteed data processing: Storm provides strong guarantees that each message passed on to it to process will be processed at least once. In the event of failures, Storm will replay the lost tuples.
- Easy to operate: Storm is simple to deploy and manage. Once the cluster is deployed, it requires little maintenance.

4.2 Spark Streaming

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Much like Spark is built on the concept of RDDs, Spark Streaming provides an abstraction called DStreams, which represents a continuous stream of data. Internally, each DStream is represented as a sequence of RDDs arriving at each time step. DStream can be created from various input sources, such as Flume [16], Kafka, HDFS, or by applying high-level operations on other DStreams.

Spark Streaming uses a "micro-batch" architecture, where the streaming computation is treated as a continuous series of batch computations on small batches of data. Spark Streaming receives data from various input sources and groups it into small batches. New batches are created at regular time intervals. At the beginning of each time interval a new batch is created. At the end of the time interval the batch is done growing. The size of the time intervals is determined by a parameter called the batch interval. The batch interval is typically between 500 ms and several seconds, as configured by the application developer. Each input batch forms an RDD, and is processed using Spark jobs to create other RDDs. The processed results are pushed out to external systems in batches. The high-level architecture is shown in Fig. 7.

Fig. 7. Spark Streaming architecture

4.3 Samza

Apache Samza is a distributed stream processing framework. Samza processes streams, which should be composed of immutable messages of a similar type or category. A Samza job is code that performs a logical transformation on a set of input streams to append output messages to set of output streams. In order to scale the throughput of the stream processor, Samza chop streams and jobs up into smaller units of parallelism: partitions and tasks.

Each partition in the stream is a totally ordered sequence of messages. Each message in this sequence has an identifier called the offset, which is unique per partition. When a message is appended to a stream, it is appended to only one of the stream's partitions. The assignment of the message to its partition is done with a key chosen by the writer.

The task is the unit of parallelism of the job, just as the partition is to the stream. Each task consumes data from one partition for each of the job's input streams. A task processes messages from each of its input partitions sequentially, in the order of message offset. There is no defined ordering across partitions. This allows each task to operate independently. The YARN scheduler assigns each task to a machine, so the job as a whole can be distributed across many machines. The number of tasks in a job is determined by the number of input partitions.

As shown in Fig. 8 Samza is made up of three layers: a streaming layer, an execution layer, a processing layer. In streaming layer, it uses Apache Kafka for messaging. Samza can better leverage Kafka's unique architectural strengths. In execution layer, it uses Apache Hadoop YARN to provide fault tolerance, processor isolation, security, and resource management. In processing layer, Samaza API provide service. Both Samza's execution and streaming layer are pluggable, and allow developers to implement alternatives if they prefer.

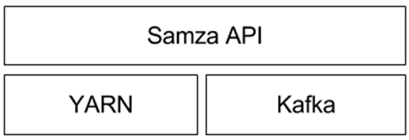

Fig. 8. Samza architecture

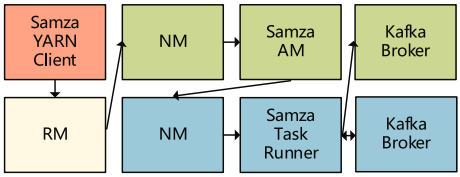

Fig. 9. Samza framework of yarn and kafka

Samza uses YARN and Kafka to provide a framework for stage-wise stream processing and partitioning. As shown in Fig. 9, everything put together, looks like this (different colors indicate different host machines).

The Samza client talks to the YARN RM when it wants to start a new Samza job. The YARN RM talks to a YARN NM to allocate space on the cluster for Samza's ApplicationMaster. Once the NM allocates space, it starts the Samza AM. After the Samza AM starts, it asks the YARN RM for one or more YARN containers to run SamzaContainers. Again, the RM works with NMs to allocate space for the containers. Once the space has been allocated, the NMs start the Samza containers.

The Samza client uses YARN to run a Samza job: YARN starts and supervises one or more SamzaContainers, and users processing code runs inside those containers. The input and output for the Samza StreamTasks come from Kafka brokers that are usually co-located on the same machines as the YARN NMs.

Samza's key features include:

- Managed state: Samza manages snapshotting and restoration of a stream processor's state. When the processor is restarted, Samza restores its state to a consistent snapshot. Samza can handle large amounts of state.
- Scalability: Samza is partitioned and distributed at every level. Kafka provides ordered, partitioned, repayable, fault-tolerant streams. YARN provides a distributed environment for Samza containers to run in.
- Fault tolerance: whenever a machine in the cluster fails, Samza works with YARN to transparently migrate your tasks to another machine.
- Durability: Samza uses Kafka to guarantee that messages are processed in the order they were written to a partition, and that no messages are ever lost.
- Processor isolation: Samza works with Apache YARN, which supports Hadoop's security model, and resource isolation through Linux CGroups (control groups).
- Pluggable: though Samza works out of the box with Kafka and YARN, Samza provides a pluggable API that lets users run Samza with other messaging systems and execution environments.

5 MPP Databases

Massive Parallel Processing (MPP) [17, 18] systems use share-nothing architecture with different computing modules coupling loosely. Unlike some other systems, MPP databases utilize a cluster of commercial x86 servers to achieve high performance, with a speed 10x or even 100x greater than traditional databases. Data are typically stored locally on each node and managed by a distributed file system. With a high compatibility with SQL, the ability to scale out and fault tolerance, MPP databases offer a cost efficient solution to big data technology.

Most MPP databases are commercial products, including Teradata, Vertica, Netezza, Amazon Redshift etc. Teradata targets high-end data warehouse and business decision-making system. However, with a rapid development of cloud computing and big data technology, the costly price of Teradata makes it less competitive today. Vertica is a column-based MPP database which offers high-speed, high-scalability and low-cost database technology. Netezza offers a warehouse solution, which combines storage, processing, database and analysis into one system. It is suitable for customers who intend to build high-level big data analysis with ready-to-use feature. As PB-level database, Amazon Redshift can easily coordinate with existing BI system. It utilizes columnar storage and data compression technology to enhance performance. Since Redshift is fully hosted at Amazon's cloud end, storage and computing resources can be assigned dynamically. Greenplum is a distributed relational database focused on

OLAP data engine. It consists of several independent small databases. Greenplum has been accepted as an open-source project by Apache Software Foundation and source code is currently hosed on GitHub.

6 NoSQL

NoSQL means "Not Only SQL", which generally refers to all non-relational databases. It is developed to handle large-scale structured, semi-structured and non-structured data brought by big data technology. NoSQL databases have several advantages such as open-source, high scalability, high concurrency, high performance, weak-transactional, and agility of development. In general, NoSQL databases can be categorized into four types, which are Key-Value type, Document-Oriented type, Column-Family Type and Graph type.

6.1 Key-Value

A Key-Value database is like a hash table used in many programming languages. One specific key is combined with one pointer, which points to some certain data. Thus, the key can be used to quickly add, query or delete data. Since data are accessed by a set of keys, such systems are able to achieve a high performance and high scalability. Redis, Memcached and other memory-based databases are normally used to in scenarios where high-speed cache on "hot data" is needed.

6.2 Document-Oriented

Document-oriented database store data as documents. Each document contains a data unit, which is a series of data collections. Each data unit contains a title and a corresponding value, which is either simple data type or complex data type. The minimum storage unit is a document. The document attribute stored in the same table can be diffident. Data can be stored as XML, JSON and JSONB, etc. MongoDB and CouchDB are the common document-oriented databases.

6.3 Column-Family

Columnar storage databases store data in column families. Frequently queried data are stored together in the same column family to handle massive data distributed on different machines. Column-Family databases typically store structured and semi-structured data. With aggressive compression technology, queries on certain columns have a great advantages on system IO. Cassandra, Hbase are the two major Column-Family databases and are used in social media websites and blogs which obtain a high volume of data in key-value type.

6.4 Graph-Oriented

Unlike other row-oriented or SQL-based databases, Graph-Oriented databases apply a more agile graph model and are able to scale out to multiple servers [19, 20]. Such

systems store data in graph where entities are considered as vertices, and the relations between entities are considered as edges [21–23]. The prevalent Graph-Oriented database is Neo4J [24, 25], which is extremely suitable for strong-relationship data such as recommendation engines.

7 Conclusion

This paper studies the typical big data analysis techniques, such as Hadoop, Spark, Storm, MPP database, and NoSQL. These technologies all have their own application scenarios. They are for batch processing, fast memory calculations, real-time computing, fast OLAP, and unstructured data, respectively. The adoption of these technologies depend on different scenarios. We have put forth some of the challenges in big data processing techniques and the areas where a lot of work can be done in future. A lot of issues concerning data parsing and data sharing still remain challenging areas of research in big data.

References

1. Jie, Z., Yao, X., Han, G.J.: A survey of recent technologies and challenges in big data utilizations. In: International Conference on Information and Communication Technology Convergence, pp. 497–499 (2015)
2. Menon, S.P., Hegde, N.P.: A survey of tools and applications in big data. In: 9th IEEE International Conference on Intelligent Systems and Control, pp. 1–7 (2015)
3. Senbalci, C., Altuntas, S., Bozkus, Z.: Big data platform development with a domain specific language for telecom industries. In: High Capacity Optical Networks and Emerging/Enabling Technologies, pp. 116–120 (2013)
4. Cho, S.Y.: Fast memory and storage architectures for the big data era. In: IEEE Asian Solid-State Circuits Conference, pp. 1–4 (2015)
5. Tseng, J.-C., Tseng, H.C., Liu, C.W.: A successful application of big data storage techniques implemented to criminal investigation for telecom. In: 2013 15th Asia-Pacific Network Operations and Management Symposium, pp. 25–27 (2013)
6. Zhang, X.X., Xu, F.: Survey of research on big data storage. In: International Symposium on Distributed Computing and Applications to Business, Engineering & Science (DCABES), pp. 76–80 (2013)
7. Yan, X., Zhang, D.: Big data research. Comput. Technol. Dev., 1–5 (2013)
8. Meng, X., Ci, X.: Big data management: concepts, technology and challenges. J. Comput. Res. Dev. **50**(1), 146–169 (2013)
9. Tan, X., Wang, H.: Big data analytics: competition and coexistence of RDBMS and MapReduce. J. Softw. **23**(1), 32–45 (2012)
10. Arun, M., Vinod, K.V.: Apache Hadoop YARN–Moving Beyond MapReduce and Batch Processing with Apache Hadoop 2. Addison-Wesley Professional, Reading (2014)
11. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Usenix Conference on Hot Topics in Cloud Computing, vol. 15, p. 10. USENIX Association (2010)

12. Armbrust, M., Xin, R.S., Lian, C., Huai, Y., Liu, D., Bradley, J.K., Meng, X., Kaftan, T., Franklin, M.J., Ghodsi, A., Zaharia, M.: Spark SQL: relational data processing in spark. In: ACM SIGMOD International Conference on Management of Data, pp. 1383–1394. ACM (2015)
13. Leibiusky, J., Eisbruch, G., Simonassi, D.: Getting Started with Storm, pp. 29–37, 39–42. O'Reilly Media, Sebastopol (2012)
14. Taylor Goetz, P., O'Neill, B.: Storm blueprints: patterns for distributed real-time computation, pp. 36–37 (2014)
15. Jain, A., Nalya, A.: Learning Storm, pp. 8–9. Packt Publishing, Birmingham (2014)
16. Karau, H., Konwinski, A., Wendell, P., Zaharia, M.: Learning Spark, pp. 182–211. O'Reilly Media, Sebastopol (2015)
17. Babu, S., Herodotou, H.: Massively parallel databases and MapReduce systems. Found. Trends Databases **5**(1), 1–104 (2012)
18. Cheng, B., Guan, X., Wu, H.: A hypergraph based task scheduling strategy for massive parallel spatial data processing on master-slave platforms. In: 23rd International Conference on Geoinformatics, pp. 1–5 (2015)
19. Xu, L., Luan, Y., Cheng, X., Cao, X., Chao, K., Gao, J., Jia, Y., Wang, S.: WCDMA data based LTE site selection scheme in LTE deployment. In: International Conference on Signal and Information Processing, Networking and Computers, Beijing, pp. 249–260. CRC Press Taylor & Francis Group (2015)
20. Xu, L., Cheng, X., Liu, Y., Chen, W., Luan, Y., Chao, K., Yuan, M., Xu, B.: Mobility load balancing aware radio resource allocation scheme for LTE-advanced cellular networks. In: IEEE International Conference on Communication Technology, Hangzhou, pp. 806–812. IEEE Press (2015)
21. Xu, L., Chen, Y., Chai, K.K., Luan, Y., Liu, D.: Cooperative mobility load balancing in relay cellular networks. In: IEEE International Conference on Communication in China, Xi'an, pp. 141–146. IEEE Press (2013)
22. Cao, Y., Sun, Z., Wang, N., Riaz, M., Cruickshank, H., Liu, X.: Geographic-based spray-and-relay (GSaR): an efficient routing scheme for DTNs. IEEE Trans. Veh. Technol. **64**(4), 1548–1564 (2015)
23. Xu, L., Luan, Y., Cheng, X., Xing, H., Liu, Y., Jiang, X., Chen, W., Chao, K.: Self-optimised joint traffic offloading in heterogeneous cellular networks. In: IEEE International Symposium on Communications and Information Technologies, Qingdao, pp. 263–267. IEEE Press (2016)
24. Xu, L., Chen, Y., Gao, Y., Cuthbert, L.: A self-optimizing load balancing scheme for fixed relay cellular networks. In: IET International Conference on Communication Technology and Application, Beijing, pp. 306–311. IET Press (2011)
25. Cao, Y., Wang, N., Sun, Z., Cruickshank, H.: A reliable and efficient encounter-based routing framework for delay/disruption tolerant networks. IEEE Sens. J. **15**(7), 4004–4018 (2015)