Big Data Management Systems for the Exploitation of Pervasive Environments

Alba Amato and Salvatore Venticinque

Abstract The amount of available data has exploded significantly in the past years, due to the fast growing number of services and users producing vast amounts of data. The Internet of Things (IoT) has given rise to new types of data, emerging for instance from the collection of sensor data and the control of actuators. The explosion of devices that have automated and perhaps improved the lives of all of us has generated a huge mass of information that will continue to grow exponentially. For this reason the need to store, manage, and treat the ever increasing amounts of data that comes via the Internet of Things has become urgent. In this context, Big Data becomes immensely important, making possible to turn into this amount of data in information, knowledge, and, ultimately, wisdom. The aim of this chapter is to provide an original solution that uses Big Data technologies for redesigning an IoT context aware application for the exploitation of pervasive environment addressing problems and discussing the important aspects of the selected solution. The chapter also provides a survey of Big Data technical and technological solutions to manage the amounts of data that comes via the Internet of Things.

1 Introduction

The amount of available data has exploded significantly in the past years, due to the fast growing number of services and users producing vast amounts of data. The Internet of Things (IoT) has given rise to new types of data, emerging for instance from the collection of sensor data and the control of actuators. The explosion of devices that have automated and perhaps improved the lives of all of us has generated

A. Amato (✉) · S. Venticinque
Department of Industrial and Information Engineering, Second University of Naples, Naples, Italy
e-mail: alba.amato@unina2.it

S. Venticinque
e-mail: salvatore.venticinque@unina2.it

a huge mass of information that will continue to grow exponentially. For this reason the need to store, manage, and treat the ever increasing amounts of data that comes via the Internet of Things has become urgent. In this context, Big Data becomes immensely important, making possible to turn into this amount of data in information, knowledge, and, ultimately, wisdom. An interesting view of what are the Big Data has been exposed to Gartner that defines Big Data as "high volume, velocity and/or variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation" [1]. In fact the huge size is not the only property of Big Data. Only if the information has the characteristics of Volume,Velocity and/or Variety we can talk about Big Data [2] as shown in Fig. 1. Volume refers to the fact that we are dealing with ever-growing data expanding beyond terabytes into petabytes, and even exabytes (1 million TB). Variety refers to the fact that Big Data is characterized by data that often come from heterogeneous sources such as machines, sensors and unrefined ones, making the management much more complex. Finally, the third characteristic, that is velocity that, according to Gartner [3], "means both how fast data is being produced and how fast the data must be processed to meet demand". In fact in a very short time the data can become obsolete. Dealing effectively with Big Data "requires to perform analytics against the volume and variety of data while it is still in motion, not just after" [2]. IBM [4] proposes the inclusion of veracity as the fourth Big Data attribute to emphasize the importance of addressing and managing the uncertainty of some types of data. Striving for high data quality is an important Big Data requirement and challenge, but even the best data cleansing methods cannot remove the inherent unpredictability of some data, like the weather, the economy, or a customer's actual future buying decisions. The need to acknowledge and plan for uncertainty is a dimension of Big Data that has been introduced as executives seek to better understand the uncertain world around them. Big Data are so complex and large that it is really difficult and sometime impossible, to process and analyze them using traditional approaches. In fact traditional relational database management systems (RDBMS) can not handle Big Data sets in a cost effective and timely manner. These technologies may not be enabled to extract, from large data set, rich information that can be exploited across of a broad range of topics such as market segmentation, user behavior profiling, trend prediction, events detection, etc and in many fields like public health, economic development and economic forecasting. Besides Big Data have a low information per byte, and, therefore, given the vast amount of data, the potential for great insight is quite high only if it is possible analyze the whole dataset [2]. In fact data is the raw material that is processed into information. Individual data by itself is not very useful, but volumes of it can identify trends and patterns. This and other sources of information come together to form knowledge. In the simplest sense, knowledge is information of which someone is aware. Wisdom is then born from knowledge plus experience [5]. So, the challenge is to find a way to transform raw data into valuable information. To capture value from Big Data, it is necessary an innovation in technologies and techniques that will help individuals and/organizations to integrate, analyze, visualize different types of data at different spatial and temporal scales. The aim of this chapter is to provide

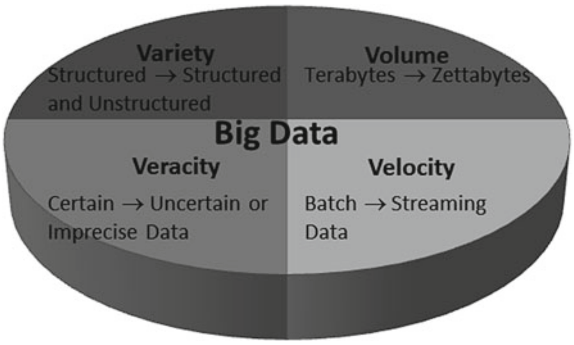

Fig. 1 Big data characteristics

an original solution that uses Big Data technologies for redesigning an IoT context aware application for the exploitation of pervasive environment addressing problems and discussing the important aspects of the selected solution. The chapter also provides a survey of Big Data technical and technological solutions to manage the amounts of data that comes via the Internet of Things. The paper is organized as follows: in Sect. 2 a survey of technical and technological solutions related to Big Data is presented, Sects. 4 and 5 describe an example of application context that is suitable for the utilization Big Data solution. Sect. 6 briefly describes the proposed solution; conclusions are drawn in Sect. 7.

2 NoSQL

The term NoSQL (meaning 'not only SQL') is used to describe a large class of databases which do not have properties of traditional relational databases and which are generally not queried with SQL (structured query language). NoSQL data stores are designed to scale well horizontally and run on commodity hardware. Also, the 'one size fit's it all' [6] notion does not work for all scenarios and it is a better to build systems based on the nature of the application and its work/data load [7].

NoSQL data stores come up with following key features [8]:

- the ability to horizontally scale simple operation throughput over many servers,
- the ability to replicate and to distribute (partition) data over many servers,
- a simple call level interface or protocol (in contrast to a SQL binding),
- a weaker concurrency model than the ACID transactions of most relational (SQL) database systems,
- efficient use of distributed indexes and RAM for data storage, and
- the ability to dynamically add new attributes to data records.

Most of the NoSQL databases, has as the main objective, the achievement of scalability and higher performance. To provide this they do not guarantee all ACID properties, but use the a relaxed set of properties named BASE:

- Basically available: Allowance for parts of a system to fail

- Soft state: An object may have multiple simultaneous values
- Eventually consistent: Consistency achieved over time.

Because the data does not have to be 100 percent consistent all the time, applications can scale out to a much greater extent. By relaxing the consistency requirement, for example, NoSQL databases can have multiple copies of the same data spread across many servers or partitions in many locations. The data is instead eventually consistent when the servers are able to communicate with one another and catch up on any updates one may have missed [9]. Proponents of NoSQL often cite Eric Brewer's CAP theorem [10], formalized in [11], which basically states that is impossible for a distributed computing system to simultaneously provide all three of the following guarantees: Consistency, Availability and Partition Tolerance (from these properties the CAP acronym has been derived). Where:

- Consistency: all nodes see the same data at the same time
- Availability: a guarantee that every request receives a response about whether it was successful or failed
- Partition Tolerance: the system continues to operate despite arbitrary message loss or failure of part of the system that create a netwrok partition

Only two of the CAP properties can be ensured at the same time. Therefore, only CA systems (consistent and highly available, but not partition-tolerant), CP systems (consistent and partition tolerant, but not highly available), and AP systems (highly available and partition tolerant, but not consistent) are possible and for many people CA and CP are equivalent because loosing in Partitioning Tolerance means a lost of Availability when a partition takes place. So, the trade-offs are complex and the NoSQL databases, acting as distributed systems, must choose between either support: AP or CP.

Common concepts in NoSQL databases are [12]:

- Sharding, also referred to as horizontal scaling or horizontal partitioning. It is a partitioning mechanism in which records are stored on different servers according to some keys. Data is partitioned in such a way that records, that are typically accessd/updated together, reside on the same node. Data shards may also be replicated for reasons of reliability and load-balancing and it may be either allowed to write to a dedicated replica only or to all replicas maintaining a partition of the data. To allow such a sharding scenario there has to be a mapping between data partitions (shards) and storage nodes that are responsible for these shards. This mapping can be static or dynamic, determined by a client application, by some dedicated 'mapping-service-component' or by some network infrastructure between the client application and the storage nodes. The downside of sharding scenarios is that joins between data shards are not possible, so that the client application or proxy layer inside or outside the database has to issue several requests and postprocess (e. g. filter, aggregate) results instead.
- Consistent hashing [13]. The idea behind consistent hashing is to use the same hash function, used to generate fixed-length output data that acts as a shortened reference to the original data, for both the object hashing and the node hashing.

This is advantageous to both objects and machines. The machines will get an interval of the hash function range and the neighboring machines can take over portions of the interval of their adjacent nodes if they leave and can assign parts of their interval if some new member node joins and gets mapped to a nearby interval. Another advantage of consistent hashing is that clients can easily determine the nodes to be contacted to perform read or write operations.

- MapReduce [14] is a programming model and an associated implementation by Google for processing and generating large data sets. Users specify a map function that processes a keyvalue pair to generate a set of intermediate keyvalue pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. MapReduce is typically used to do distributed computing on clusters of computers. When applied to databases, MapReduce processes a set of keys by submitting the process logic (map- and reduce-function code) to the storage nodes which locally apply the map function to keys that should be processed and that are in their set. The intermediate results can be consistently hashed just as regular data and processed by the following nodes in clockwise direction, which apply the reduce function to the intermediate results and produce the final results. It should be noted that due to the consistent hashing of the intermediate results there is no coordinator needed to direct the processing nodes to find the intermediate results. A popular open source implementation is Apache Hadoop [15], a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage and to execute queries and other batch read operations against massive datasets that can be tens or hundreds of terabytes and even petabytes in size. A commercial closed source model is Dryad [16]. Originated by Microsoft Research, Dryad, DryadLINQ and the Distributed Storage Catalog (DSC) are currently available as community technology previews. DryadLINQ allows programmers to use Microsoft technologies such as Microsoft .NET and LINQ to express their algorithms. Dryad is a general purpose runtime for execution of data parallel applications allowing the execution of the algorithms across large quantities of unstructured data distributed across clusters of commodity computers. DSC is a storage system that provides the bottom layer to the stack. It ties together a large number of commodity machines to store very large (i.e., Bing-level) quantities of data. These are commercial versions of the same technology used by the Bing search engine for large, unstructured data analysis.
- Versioning of Datasets in Distributed Scenarios. If Datasets are distributed among nodes, they can be read and altered on each node and no strict consistency is ensured by distributed transaction protocols. Questions arise on how concurrent modifications and versions are processed and to which values a dataset will eventually converge to. There are several options to handle these issues and, the most used are Multiversion Concurrency Control (MVCC), whose aim is to avoid the problem of writers blocking readers and viceversa, by making use of multiple versions of data, and vector clocks, an algorithm for generating a partial ordering of events in a distributed system and detecting causality violations.

However, NoSQL databases are not the solution to every problem of data management [17]. In fact, they completely miss a common query language like SQL in RDBMS. SQL is based on Relational Algebra that ensures completeness of the query language and that offers many optimization techniques to support query execution. It represents one of the main reasons why the RDBMS systems have acquired increasing importance. A developer can move from one database system to another with reduced effort at least for basic operations. Another lack of NoSQLs is the extreme heterogeneity of the existing solutions for what concerns the organization of the data model, the query model and the data access recommended patterns. This forces the developer to handle manually low-level data management issues like indexing, query optimizing data structures, relations between objects, and so on. This results in a higher complexity of NoSQL compared to RDBMS solutions for what concerns programmability and management of the data store.

3 NoSQL Classification

According to [8] NoSQL Data Models can be classified in:

- Key-value data stores (KVS). They store values associated with an index (key). KVS systems typically provide replication, versioning, locking, transactions, sorting, and/or other features. The client API offers simple operations including puts, gets, deletes, and key lookups. Notable examples include: Amazon DynamoDB, Project Voldemort, Memcached, Redis and RIAK
- Document data stores (DDS). DDS typically store more complex data than KVS, allowing for nested values and dynamic attribute definitions at runtime. Unlike KVS, DDS generally support secondary indexes and multiple types of documents (objects) per database, as well as nested documents or lists. Notable examples include Amazon SimpleDB, CouchDB, MembaseCouchbase, MongoDB and RavenDB
- Extensible record data stores (ERDS). ERDS store extensible records, where default attributes (and their families) can be defined in a schema, but new attributes can be added per record. ERDS can partition extensible records both horizontally (per-row) or vertically (per-column) across a datastore, as well as simultaneously using both partitioning approaches. Notable examples include Google BigTable, HBase, Hypertable and Cassandra

Another important category is constituted by Graph data stores. They [18] are based on graph theory and use graph structures with nodes, edges, and properties to represent and store data. Key-Value, Document based and Extensible record categories aim at the entities decoupling to facilitate the data partitioning and have less overhead on read and write operations, whereas Graph-based take the modeling the relations like principal objective. Therefore techniques to enhancing schema with a Graph-based database may not be the same as used with Key-Value and others. The graph data model fits better to model domain problems that can be represented by graph as ontologies, relationship, maps etc. Particular query languages allow querying the data bases by using classical graph operators as neighbor, path, distance etc. Unlike the NoSQL systems we presented, these systems generally provide ACID transactions. Different Graph Store products exist in the market today. Some provide custom API's and Query Languages and many support the W3C's RDF standard. Notable examples include neo4j, AllegroGraph and InfiniteGraph.

Amazon Dynamo [19] is a key-value distributed storage system that is developed and used by Amazon. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance. Dynamo is a structured overlay based on consistent hashing with maximum one-hop request routing. Dynamo uses a synthesis of well known techniques to achieve scalability and availability: Data is partitioned and replicated using consistent hashing, and consistency is facilitated by object versioning [20]. The consistency among replicas during updates is maintained by a quorum-like technique and a decentralized replica synchronization protocol. Dynamo employs a gossip based distributed failure detection and membership protocol. Dynamo is a completely decentralized system with minimal need for manual administration. Storage nodes can be added and removed from Dynamo without requiring any manual partitioning or redistribution.

Project Voldemort [21] is an open source distributed key-value data store used by LinkedIn for high-scalability storage. It represents an open source implementation of the basic parts of Dynamo's distributed key-value storage system. Like Dynamo, Voldemort uses consistent hashing for data partitioning and supports virtual nodes. Data is automatically replicated over multiple servers and is automatically partitioned so each server contains only a subset of the total data. Server failure is handled transparently. Pluggable serialization is supported to allow for rich keys and values including lists and tuples with named fields, as well as to integrate with common serialization frameworks like Protocol Buffers, Thrift, Avro and Java Serialization. Data items are versioned to maximize data integrity in failure scenarios without compromising availability of the system and each node is independent of other nodes with no central point of failure or coordination. It also supports pluggable data placement strategies to support geographically separated datacenters. Voldemort single node performance is at the range of 10–20k operations per second depending on the machines, the network, the disk system, and the data replication factor. Voldemort provides eventually consistency, just like Amazon Dynamo [22].

Memcached [23] is a free and open source, high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load. Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering. Keys are hashed and ash table span across an arbitrary number of servers. Its API is available for most popular languages.

Redis [24] Redis is an open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets. It is possible to run atomic operations on these types, like appending to a string; incrementing the value in a hash; pushing to a list; computing set intersection, union and difference; or getting the member with highest ranking in a sorted set. In order to achieve better performance, Redis works with an in-memory dataset and it is possible to persist it either by dumping the dataset to disk every once in a while, or by appending each command to a log. Redis in particular does not offer fault-tolerance and as data is held in memory it will be lost if a server crashes.

RIAK [25] is another open source distributed key-value data store, developed by Basho Technologies, that provides tunable consistency. Consistency is tuned by specifying how many replicas must respond for a successful read/write operation and can be specified per-operation. It provides a decentralized key-value store that supports standard Get, Put and Delete operations. Riak is a distributed, highly scalable and fault-tolerant store with map/reduce, HTTP, JSON and REST queries. RIAK relies on consistent hashing for data partitioning and vector clocks for versioning, like Dynamo and Voldemort. Riak also includes a MapReduce mechanism for non-key-based querying. MapReduce jobs can be submitted through the RIAK's HTTP API or the protobufs API. To this end, the client makes a request to RIAK node which becomes the coordinating node for the MapReduce job.

Amazon SimpleDB [26] is a proprietary document data store offered as a service in Amazon's AWS cloud portfolio. Amazon SimpleDB automatically manages infrastructure provisioning, hardware and software maintenance, replication and indexing of data items, and performance tuning. Amazon SimpleDB automatically creates multiple geographically distributed copies of each data item you store. This provides high availability and durability in the unlikely event that one replica fails, Amazon SimpleDB can failover to another replica in the system. Amazon SimpleDB supports two read consistency options: eventually consistent reads and consistent reads. The eventual consistency option (default) maximizes the read performance (in terms of low latency and high throughput). However, an eventually consistent read might not reflect the results of a recently completed write. Consistency across all copies of data is usually reached within a second; repeating a read after a short time should return the updated data. So in addition to eventual consistency, Amazon SimpleDB also gives to the user the flexibility and control to request a consistent read if the application requires it. A consistent read returns a result that reflects all writes that received a successful response prior to the read. Unlike with key-value datastores, SimpleDB supports more than one grouping in one database: documents are put into domains, which support multiple indexes. SimpleDB data model is comprised of domains, items, attributes and values. Domains are collections of items that are described by attribute-value pairs. SimpleDB constrains individual domains to grow up to 10 GB each, and currently has a limit of 100 active domains.

Apache CouchDB [27] is an open source database that focuses on ease of use and on being "a database that completely embraces the web". It stores JSON objects that consist of named fields without predefined schema. Field values can be strings, numbers, or dates; but also ordered lists and associative arrays. CouchDB uses JavaScript for MapReduce queries, and regular HTTP for an API and provides ACID semantics at the document level, but eventual consistency otherwise. To support ACID on document level, CouchDB implements a form of Multi-Version Concurrency Control (MVCC) in order to avoid the need to lock the database file during writes. Conflicts are left to the application to resolve. CouchDB structure stores data into views and each view is constructed by a JavaScript function that acts as the map phase in MapReduce. CouchDB was designed with bi-directional replication (or synchronization) and off-line operation in mind. Namely, CouchDB can replicate to devices (like smart-phones) that can go offline and later sync back the device.

Couchbase [28], originally known as Membase, is an open source, distributed document oriented data store that is optimized for interactive applications. These applications must serve many concurrent users; creating, storing, retrieving, aggregating, manipulating and presenting data. In support of these kinds of application needs, Couchbase is designed to provide easy-to-scale key-value or document access with low latency and high sustained throughput. It is designed to be clustered from a single machine to very large scale deployments. Couchbase has initially grown around memcached, by adding to it features like persistence, replication, high availability, live cluster reconfiguration, re-balancing, multi-tenancy and data partitioning. Couchbase supports fast fail-over with multi-model replication support for both peer-to-peer replication and master-slave replication. Every Couchbase node is architecturally identical consisting of a data manager and cluster manager component. The cluster manager supervises the configuration and behavior of all the servers in a Couchbase cluster. It configures and supervises internode behavior like managing replication streams and rebalancing operations. It also provides metric aggregation and consensus functions for the cluster, and a RESTful cluster management API. The cluster manager is built atop Erlang/OTP, a proven environment for building and operating fault-tolerant distributed systems. The data manager is responsible for storing and retrieving documents in response to data operations from applications. Couchbase has only recently migrated from a key-value store to a document data store, with version 2.0 bringing features like JSON document store, incremental MapReduce and cross datacenter replication. Couchbase stores JSON objects with no predefined schema.

MongoDB [29], is an open source document-oriented data storebase system. MongoDB stores structured data as JSON-like documents with dynamic schemas. MongoDB supports queries by field, range queries, regular expression searches. Queries can return specific fields of documents and also include user-defined JavaScript functions. Any field in a MongoDB document can be indexed and secondary indices are also available. MongoDB supports master-slave replication. A master can perform reads and writes, whereas a slave copies data from the master and cannot be used for writes. MongoDB scales horizontally using sharding and can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure. MongoDB supplies a file system function, called GridFS, taking advantage of load balancing and data replication features over multiple machines for storing files. MapReduce can be used in MongoDB for batch processing of data and aggregation operations.

RavenDB [30], is a transactional, open source Document Database written in .NET, and offering a flexible data model designed to address requirements coming from real-world systems. RavenDB allows you to build high-performance, low-latency applications quickly and efficiently. Data in RavenDB is stored schema-less as JSON documents, and can be queried efficiently using Linq queries from your .NET code or using RESTful API using other tools. Internally, RavenDB makes use of indexes which are automatically created based on your usage, or created explicitly by the consumer. RavenDB is built for web-scale, and offers replication and sharding support out-of-the-box.

Big Table [31], Bigtable is a distributed storage system designed by Google for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Google initially designed BigTable as distributed data storage solution for several applications (like Google Earth and Google Finance), aiming at providing flexible, high-performance solution for different application requirements. It is designed for storing items such as billions of URLs, with many versions per page; over 100 TB of satellite image data; hundreds of millions of users; and performing thousands of queries a second. Bigtable is designed with semi-structured data storage in mind. It is a large map that is indexed by a row key, column key, and a timestamp. Each value within the map is an array of bytes that is interpreted by the application. Every read or write of data to a row is atomic, regardless of how many different columns are read or written within that row.

Hbase [32] is a distributed, column-oriented, data storage system offering strict consistency designed for data distributed over numerous nodes. It provides strongly consistent reads and writes versus an eventually consistent data store. HBase is largely inspired by Google's BigTable and is designed to work well with Hadoop which is an open source implementation of Google's MapReduce framework. The default distributed files system for Hadoop (HDFS) is designed for sequential reads and writes of large files in a batch manner. This strategy disallows the system to offer close to real-time access which requires efficient random accesses of the data. HBase is an additional layer on top of HDFS that efficiently supports random reads—and in general access—on the data, using a sparse multi-dimensional sorted map. HBase does not support a structured query language like SQL. HBase applications are written in Java much like a typical MapReduce application. HBase does support writing applications in Avro, REST, and Thrift.

Hypertable [33] is a high performance, open source, massively scalable database modeled after Bigtable, Google's proprietary, massively scalable database. Its goal is to set the open source standard for highly available, scalable, database systems. Hypertable runs on top of a distributed file system such as the Apache Hadoop DFS, GlusterFS, or the Kosmos File System (KFS). It is written almost entirely in C++. Hypertable is similar to a relational database in that it represents data as tables of information, with rows and columns, but the main differences are that row keys are UTF-8 strings, there is no support for data types, values are treated as opaque byte sequences. Regarding its data model it supports all abstractions available in Bigtable; in contrast to Hbase column-families with an arbitrary numbers of distinct columns are available in Hypertable. Tables are partitioned by ranges of row keys (like in Bigtable) and the resulting partitions get replicated between servers. The data representation and processing at runtime is also borrowed from Bigtable; updates are done in memory and later flushed to disk. Hypertable has its own query language called HQL (Hypertable Query Language) and exposes a native C++ as well as a Thrift API.

Cassandra [34] is a distributed data storage system developed by Facebook which, similarly to BigTable, is designed for managing very large amounts of structured data spread out across many commodity servers, providing a key-value store with tunable consistency. Cassandra lets the application developer dial in the appropriate level of consistency versus scalability or availability for each transaction. This tunable consistency is a level of sophistication that other databases such as HBase do not always offer. However, the extra sophistication comes with more of a learning curve for the developer [9]. The main goal of Cassandra is to provide a highly available service with no single point of failure. The Cassandra API [35] consists of three very simple methods (insert, get, delete) and it allows the user to manipulate data using a distributed multi-dimensional map indexed by the key. The different attributes (columns) of the data stored by Cassandra are grouped together into sets (called "column families"). Cassandra exposes two kinds of such families: "simple column families" and "super column families", where the latter are a column family within a column family. This allows a key to map to multiple values. Cassandra also has an in-memory cache to speed access to the most important data on designated servers in the cluster. In terms of scalability, Cassandra achieves the highest throughput for the maximum number of nodes in all experiments [36].

Neo4j [37], is an open source, robust (fully ACID) transactional property graph database. Due to its graph data model, Neo4j is highly agile and blazing fast. For connected data operations, Neo4j runs a thousand times faster than relational databases. Nodes store data and edges represent relationships. The data model is called property graph to indicate that edges could have properties. Neo4j provides a REST interface or a Java API. The core engine of Neo4j supports the property graph model. This model can easily be adapted to support the LinkedData RDF model, consisting of Triples. Besides it is possible to add spatial indexes to already located data, and perform spatial operations on the data like searching for data within specified regions or within a specified distance of a point of interest. In addition classes are provided to expose the data to geotools and thereby to geotools enabled applications like geoserver and uDig.

AllegroGraph [38] is a modern, high-performance, persistent graph database. AllegroGraph uses efficient memory utilization in combination with disk-based storage, enabling it to scale to billions of quads while maintaining superior performance. AllegroGraph supports SPARQL, RDFS++, and Prolog reasoning from numerous client applications. AllegroGraph is a proprietary product of Franz Inc., which markets a number of Semantic Web products and claims Pfizer, Ford, Kodak, NASA and the Department of Defense among its AllegroGraph customers.

InfiniteGraph [39] InfiniteGraph is a proprietary graph database currently available in both free and paid license versions produced by Objectivity, a company that develops data technologies supporting large-scale, distributed data management, object persistence and relationship analytics. Its goal is to create a graph database with "virtually unlimited scalability". InfiniteGraph is used in applications including real-time and location-aware web and mobile advertising platforms, military

Table 1 Data store comparison

Name	Classification	License	Data storage
Dynamo	KVS	Proprietary	Plug-in
Voldemort	KVS	Open source	RAM
Memcached	KVS	Open source	RAM
Redis	KVS	Open source	RAM
RIAK	KVS	Open source	Plug-in
SimpleDB	DDS	Proprietary	S3 (simple storage service)
CouchDB	DDS	Open source	Disk
Couchbase	DDS	Open source	Disk
MongoDB	DDS	Open source	Disk
RavenDB	DDS	Open source	Disk
Google BigTable	ERDS	Proprietary	GFS
HBase	ERDS	Open source	Hadoop
Hypertable	ERDS	Open source	Disk
Cassandra	ERDS	Open source	Disk
Neo4J	Graph	Open source	Disk
AllegroGraph	Graph	Proprietary	Disk
InfiniteGraph	Graph	Proprietary	Disk

operations planning and mission assurance, and advanced healthcare and patient records management.

Table 1 provides a comparison of all the examples given in terms of Classification, Licence and Storage System. Comparison based on several issues are available at [40].

4 Context Awareness in Pervasive Environments

Pervasiveness of devices provides to application and services the possibility for using peripherals and sensors as their own extensions to collect about the user and the environment, but also to improve service delivery. In the last years, context awareness has widely demonstrated its crucial role to achieve optimized management of resources, systems, and services in many application domains, from mobile and pervasive computing to dynamically adaptive service provisioning. Furthermore the explosion of devices that have automated and perhaps improved the lives of all of us has generated a huge mass of information that will continue to grow exponentially. In fact if we take in consideration, that by 2020 there will be more than 30 billion devices connected to the Internet, it is possible to understand how much data this devices could produce if they are connected 24/7. Data can be collected from various sources such as social networks, data warehouse, web applications, networked machines, virtual machines, sensors over the network, mobile devices, etc. It is necessary to think about how and where to store them. It is necessary a scalable, distributed storage systems, a set of flexible data models that allow for an easy interaction with programming languages. The need to store, manage, and treat the ever increasing amounts of data is becoming increasingly felt and contextualisation can be an attractive paradigm to combine heterogeneous data streams to improve quality of a mining process or classifier. Another issue concerns the integration of multiple data sources in an automated, scalable way to aggregate and store these heterogeneous and unbelievable amounts of data to conduct deep analytics on the combined data set. The traditional storage can represent a low cost solution to handle this kind of information. Also the effort spent in redesigning and optimizing data warehouse for analysis requests could result in poor performance. In fact current databases and management tools are inadequate to handle complexity, scale, dynamism, heterogeneity, and growth of such systems.

For these reasons context awareness in pervasive environments represent an interesting application field of big data technologies. In fact, according to IBM [2]:

- Big Data solutions are ideal for analyzing not only raw structured data, but semi-structured and unstructured data from a wide variety of sources.
- Big Data solutions are ideal when all, or most, of the data needs to be analyzed versus a sample of the data; or a sampling of data is not nearly as effective as a larger set of data from which to derive analysis.
- Big Data solutions are ideal for iterative and exploratory analysis when measures on data are not predetermined.

Big data technologies can address the problems related to the collection of data streams of higher velocity and higher variety. They allow for building an infrastructure that delivers low, predictable latency in both capturing data and in executing short, simple queries; that is able to handle very high transaction volumes, often in a distributed environment; and supports flexible, dynamic data structures [41]. With such a high volume of information, it is relevant the possibility to organize data at its original storage location, thus saving both time and money by not moving around large volumes of data. The infrastructures required for organizing big data are able to process and manipulate data in the original storage location. This capability provides very high throughput (often in batch), which are necessary to deal with large data processing steps and to handle a large variety of data formats, from unstructured to structured [41]. The analysis may also be done in a distributed environment, where some data will stay where it was originally stored and be transparently accessed for required analytics such as statistical analysis and data mining, on a wider variety of data types stored in diverse systems; scale to extreme data volumes; deliver faster response times driven by changes in behavior; and automate decisions based on analytical models. Most importantly, the infrastructure must be able to integrate analysis on the combination of big data and traditional enterprise data. New insight comes not just from analyzing new data, but from analyzing it within the context of the old to provide new perspectives on old problems [41]. Context-aware Big Data solutions could focus only on relevant information by keeping high probability of hit for all application-relevant events, with manifest advantages in terms of cost reduction and complexity decrease [42].

5 The M.A.R.A. Case Study

Exploitation of archaeological sites can be very difficult because of a lack of supporting infrastructures and because of the complex recognition and comprehension of the relevant ruins, artworks and artifacts. The availability of personal devices can be used to plan the visit and to support the tourist by suggesting him the itineraries, the points of interest and by providing multimedia contents in the form of digital objects which can semantically augment the perceived reality. In this context relevant issues are the profiling of the user, the discovery and the delivery of the contents that can improve the user's satisfaction, new models of interactions with reality. The Second University of Naples is engaged on a multidisciplinary project with both cultural and a technological aims [43]. Three case studies have been chosen to test the approach and the framework. The S. Angelo in Formis Basilica, in Campania, near S. Maria Capua Vetere, the ancient town of Norba and the amphitheater of Capua. In these sites we cannot install complex infrastructures and they are difficult to understand without a tourist guide. For the presented case studies we need to provide a technological solution that does not need infrastructures for letting the software know the user location and his feeling about the environment [44]. It means that Bluetooth, NFC, GPS, electronic compass, camera, network connection and others technologies have to be used, together or independently, to get information about the user perceptions and to augment his exploitation of the archaeological site. In particular we will deal with context aware recommender systems that are achieving widespread success in nowadays in a lot of fields. The aim of those systems is making personalized recommendations during a live user interaction. A recommendation system learns from user's behavior and makes recommendation that can be interesting for users. The key component of a context aware recommendation system is data, often extremely diverse in format, frequency, amount and provenance. This heterogeneous data will serve as the basis for recommendations obtaining using algorithms to find similarities and affinities and to build suggestions for specific user profiles. One of the most important application field of recommender systems is cultural heritage. Archaeological sites become pervasive environments if personal devices like tablets and smart-phones are able to detect surrounding ruins, artifact and other kind of points of interest by their on-board peripherals. In this context pervasiveness offers to software applications the possibility to interact with the reality by the device, in order to perceive the information surrounding the users, and to adapt their own behavior and the environment itself. By modeling an archaeological site as a pervasive environment we are able to improve its exploitation by the visitors. In a pervasive computing environment, the intelligence of context-aware systems will be limited if the systems are unable to represent and reason about context. In domains like tourism, the notion of preferences varies among users and strongly depends on factors like users' personalities, parameters related to the context like location, time, season, weather and others elements like user's feedback, so it is necessary to provide users with many other kinds of personalized experiences, based on data of many kinds. The growth of visitors and interactive archaeological sites in recent years poses some key challenges for recommender systems. In fact it is necessary to introduce recommender system technologies that can quickly produce high quality recommendations, even for very large-scale problems, so that users can benefit of context awareness in services exploitation and mobile services can became really useful and profitable. Besides, it is necessary to address the problem of the variety of data from sensors, RFID, devices, annotation tools, GIS (Geospatial, GPS), web. So the problem is both to capture data quickly and to store them quickly in structured form. The structure of the data, then, allows to identify a pattern based strategy for the extraction of consistent, comparable and updated information. In Fig. 2 an high level representation of our general case study is shown. We model the environment where the user is moving and to reconstruct the perceptions of the user himself in order to get his particular vision about what is surrounding him. A real representation of the environment is necessary to identify landmarks and possibilities of intervention using pervasive actuators and sensors whose input will be updated as the environment changes. The environment has been modeled as a geo-referred map with itineraries, landmarks and point of interest. The user can download the map of the area to be visited at home, before to leave, or on site, if the network will be available. The map includes all the points of interest that identify the relevant objects of that area and different cultural itineraries which could be exploited on site. Also contents can be discovered and downloaded in advance according to the device capabilities. Software on user's mobile device and remote services will assist the cultural visit by augmenting the reality by the user's personal device. A software agent executes on the user's device to support services exploitation. It percepts the surrounding environment using the on-board peripherals and executes plans which are chosen by an ad-hoc reasoning to optimize the user's satisfaction. The knowledge of the environment acquired by the agent represents part of its own beliefs. Another set of believes describes the user profile by recording and evaluating hiher actions or explicitly asking feedbacks. Some examples are user's position, interest, nearby objects, landscape, etc. Of course the way of localization of users and objects depends on the device technology, the available infrastructures and the kind of environment. Indoor or outdoor localization can be implemented using heterogeneous technologies, and often absolute localization can not be performed, but it is only possible to detect nearby landmarks or objects [45]. These techniques can result quite intensive in terms of computational requirements, and so the needed resources can exploit a distributed infrastructure. Part of the computation will be performed locally on the smartphone and expensive tasks will be off-loaded remotely. Besides the limited energy and data storages of the user's device can affect agent's capability. Remote services allows to the agent to move on remote more complex reasoning on a wider knowledge base. In order to augment user's knowledge and capability to interact with the environment, services have to choose, according to their context awareness; (1) what content and application it has to deliver; (2) when it needs to present the content; (3) how this should be done.

In Fig. 3 the architectural solution of the MARA framework is shown. The framework is composed of different tools and applications [44]. Tools for experts in the domain of the Cultural Heritage are used to augment the archaeological site with a set of multimedia contents. They include a map editor, a semantic annotator and a content

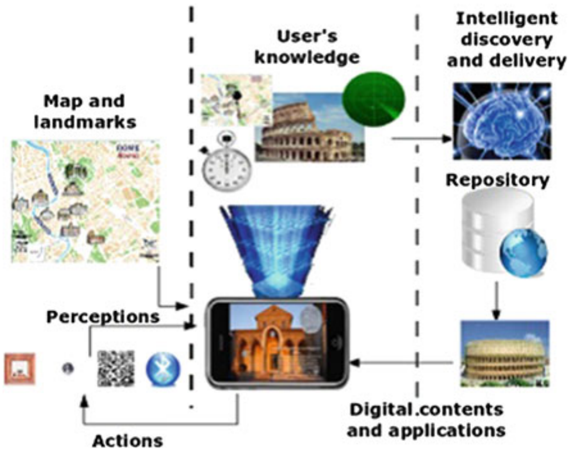
User's knowledge
Intelligent discovery and delivery
Map and landmarks
Repository
Perceptions
Actions
Digital.contents and applications

Fig. 2 Problem model

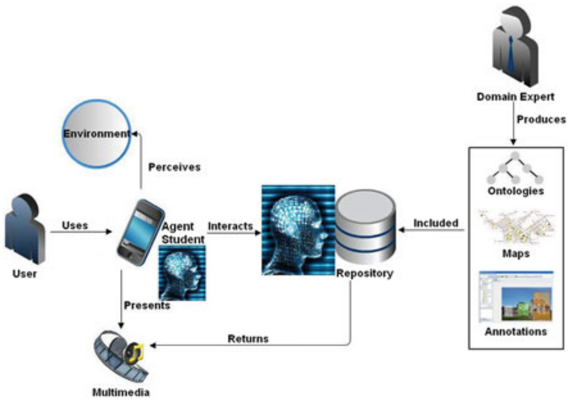

Fig. 3 Architecture and roles

manager. A set of context aware services are available for intelligent multimedia discovery and delivery. A tourist guide supports the user in visiting an archaeological site, detects and suggests points of interest, provides context awareness of remote service, allows for the utilization of remote services and plays multimedia contents. On the left side of Fig. 3, the user is using his device that hosts a light agent that is able to perceive information from the field by pervasive sensors. The agent executes autonomously and proactively in order to support the user's activity within the environment where he is moving. It discovers surrounding objects, it uses them to update the representation of the user's knowledge, reacts using the local knowledge to organize and propose the available contents and facilities by an interactive interface. If the connection works the device can access remote services, which can access a wider knowledge and have greater reasoning capabilities to look for additional contents and applications. Experts of the application domain define the ontology for the specific case study described in [46]. They use or design a map to represent the environment.

They add POIs to the map to geo-refer multimedia contents and can link them to a concept of the ontology. Furthermore they select relevant contents and annotate them using concepts and individuals of the ontology. Remote applications implement context aware services. They use personal devices to collect perceptions and for content delivery. An ontology implements the representation of the global knowledge that is necessary to share a common dictionary and to describe the relationships among the entity eobjects, which are part of the model. In our model a common ontology include all the general concepts, which are useful for a description of a pervasive environment where mobile users are moving, using their devices and interacting with available facilities and other users. The general ontology is complemented with a domain ontology that is designed by an expert of the specific application field. Concepts of the ontology are used on client side to describe a representation of the reality as it is perceived by the user. On the back-end the ontology is used to annotate digital resources like point of interests, contents, applications. It is also used to support reasoning. User's behaviors, information from pervasive devices or from other users, device properties, external events are heterogeneous data that are perceived by the device and that are used to build a dynamic changing representation of the user knowledge about the reality, within which he is moving. The applications are knowledge driven. The user's knowledge can be used by the application that is running on the device to adapt its logic locally, an is updated remotely to improve the awareness of services at server side. Application are events based. Events are triggers for reactive activity by the agent. An event may update beliefs, trigger plans or modify goals. Events may be generated externally and received by sensors or integrated systems. Additionally, events may be generated internally to trigger decoupled updates or plans of activity. Events can be updates of the user's knowledge or can be explicit service requests raised by the user. At each invocations semantic queries, that depend on the user's knowledge, are built and processed to get the action to be performed and the contents to be delivered. Results of the query are individuals of the ontology that are described by semantic annotation. The user's knowledge is composed of many semantic concepts with *static* and *dynamic* properties. Semantic techniques are used for intelligent content and application discovery and delivery. An ontology has been designed to describe the sites of interest and to annotate the related media. A general part includes the concepts which are common to all the class of applications that can be modeled according to the proposed approach. Among the others the *Time* class and his properties (*CurrentTime*, *AvailableTime*, *ElapsedTime*, *Exploitation-Time*) allow to organize and assist the visit taking into account time information and handling time constraints. *Position* class and its properties allow to localize the user and objects around him. An application specific part of the ontology includes the concepts that belong to the domain of the cultural heritage and additional classes and individual which are proper of the case studies introduced in the previous section. The ontology is used also for annotating the multimedia contents. To annotate texts, images and any kind of contents we chose the AktiveMedia tool. In Fig. 4 a picture of the Amphitheater of S. Maria Capua Vetere is annotate with the *Column* and the *Arc* classes which are part of this kind of building. The perceptions are automatically communicated by a client application for Android Device and recommendations are

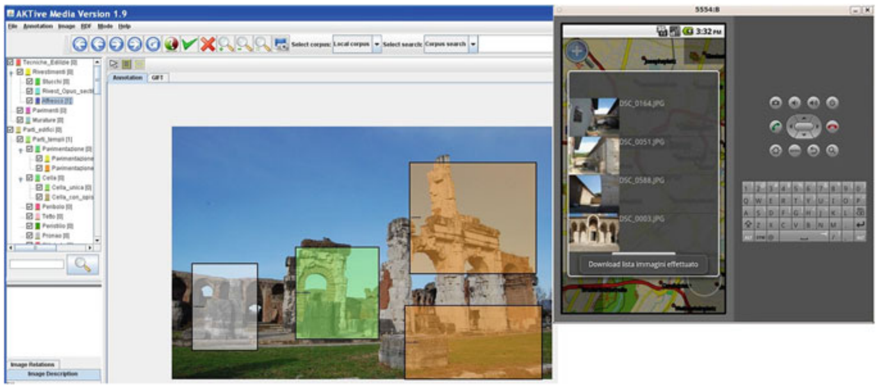

Fig. 4 The annotator

asincronously notified to the user by the tourist guide that periodically search for the most relevant items. The output produced by the annotator is an RDF file with concepts and properties of the AktiveMedia ontology and of the domain ontology.

5.1 Digital Repository and Semantic Discovery

The Fedora repository is used to store digital objects and supports their retrieval. Into the Fedora repository a digital object is composed of a set of files which are:

- *object metadata*: used by the client application to understand how to deliver the content;
- *binary streams*: which are images, video, text ... any kind of raw information to be delivered;
- *RDF annotation*: that describe the semantic of the object according to the ontology;
- *disseminations*: filters to be eventually used for adapting the object according to the target client.

We loaded the Aktive-Media ontology and the domain ontology into the Fedora repository in order to exploit its embedded SPARQL engine that is used to select the optimal set of individuals (i.e. contents). Multimedia contents are automatically stored into the repository after the annotation phase. The RDF output is automatically processed using an XSL transformation to make it compliant with the model used by the Fedora repository. Each content can be linked position.

5.2 Content Types

Different types of content models have been defined and simple examples have been produced.

- Multiple images whose transparency can be graduated by the user to compare changes in different periods. In the same way real picture can be compared with paintings. Old picture can be compared with what is seen by the camera.
- Part of the image acquired by the camera are recognized and linked to related multimedia contents.
- Virtual reconstructions which are synchronized with the camera output or the detected RFIDs.
- Text, audio, video and composite media.

A content descriptor is attached to every digital object. It is used by the device when the content is being delivered. The descriptor defines the right player for that media, configuration parameter and necessary input.

The semantic discovery service of MARA returns a set of digital objects related to POIs in the pervasive environments. Each content is annotated by concepts from the ontology and can be discovered by a SPARQL query to the content repository. The result of the query is a set of N instances of digital objects whose relevance to the user context is calculated as described in [47].

6 Moving to Big Data

Performance figures discussed in [48] demonstrate feasibility of the proposed solution implemented with a classical RDBMS. However a number of limitations have been assumed. First of all the amount of data are limited to the proprietary knowledge base with a limited number of archaeological sites. If we aim at handling all the national cultural heritage or the world one volume of data will be not supported. Besides the coverage of an increasing number of sites, eventually wider, will affect the amount of geographical information and the number of connected mobile users. Data continuously received from thousands of devices scattered in the environment handling queries and providing perception will augment volume, velocity and variety of information. Finally the exploitation of the user feedback could be used to improve the expertise of the system building a social network of visitors and enriching the knowledge base with semantic annotation inferred by a social network of visitors who become authors and editors themselves. The new vision of M.A.R.A. could not be implemented without considering the Big Data requirements and solutions that are ideal [2] to deal with raw data from a wide variety of sources that must be analyzed in toto. At this point we wonder understand what would be the best choice for re-designing and developing the framework to satisfy the new requirements. Understanding what NoSQL data models and technology could be more effective among the available alternatives needs further insides.

First of all maps, contents and users are represented by documents and profiles which are structured XML documents, but containing attributes like concepts, keyvalue pairs, RDF triples, which can dynamically change.

In current implementation of the MARA knowledge base all this documents are indexed and stored in a relational databases. Ontology, annotations, maps and profiles are processed when they are uploaded and updated by new inserting new record or updating the existing one. For this reason the adoption of a document data store supporting the map reduce paradigm seems the most rational alternative. I would not impact on the original design and supports the distribution of both the data and the computational effort for indexation and retrieval. Also users and data locality could be exploited to optimize performance. In fact points of interest and contents close to the users will be eventually more relevant to the others. About the drawback of such a choice, we have to consider that NOSQL DDS usually do not support ACID transactions, than there is the lack of a SPARQL interface for reasoning and semantic retrieval of relevant information, which is currently used by the MARA application layer. Furthermore some technological solution limit the maximum allowed size for a digital object. In fact MARA uses ontologies defined using RDFOWL that can be naturally represented using a labeled, directed, multi-graph. RDF [49] extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications. This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes. This graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations. To enables inferences and as query language, is used SPARQL that is a subgraph pattern matching query language. Besides, in MARA project we need to integrate multiple independent schemas that evolve separately depending on the different conditions correlated to the archaeological site. Semantic terms link instance data with other resources and apps and linked resources enable interoperability between apps as shown in Fig. 5. As an additional factor, it is necessary to bridge the structured and unstructured worlds and to make queries that integrate across data. It is a complex task because the different searches need to follow links between the data elements across schemas to find relationships. Graph databases [18] may overcome these limitation of the as they generally provide ACID transactions. Adoption of such solution will provide custom API's and Query Languages and many support the W3C's RDF standard, including a SPARQL engine. This model can easily integrated in MARA paltform as it supports the LinkedData RDF model. Besides it is possible to add spatial indexes to already located data, and perform spatial operations on the data like searching for data within specified regions or within a specified distance of a point of interest. In addition classes are provided to expose the data to geotools and thereby to geotools enabled applications like geoservers. As shown in Fig. 6, the second choice requires the re-desing of the knowledge base and of applications and tools for production, collection and indexing of data. In fact the new knowledge base can be modeled as a unique ontology dynamically augmented and updated with new information.

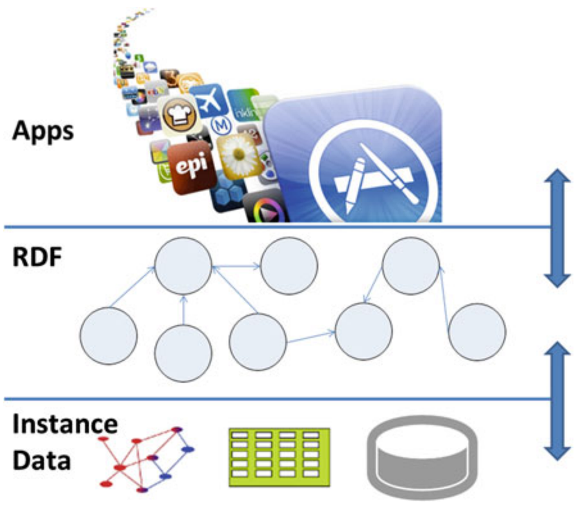

Fig. 5 RDF utilization

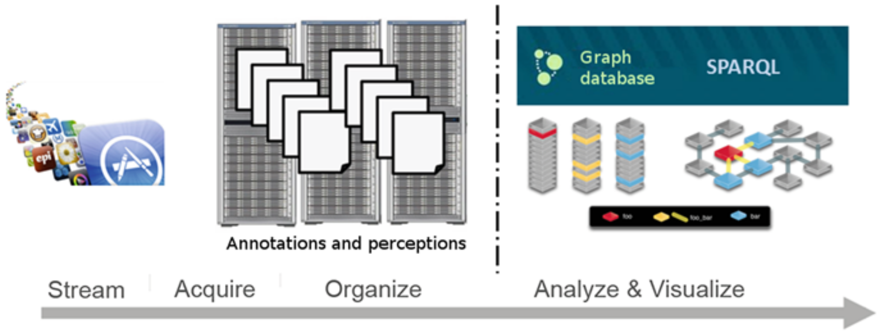

Fig. 6 Graph database utilizzation

7 Conclusion

In this chapter we have presented a survey of technical and technological solutions and analyzed critically the utilization of Big Data to redesign a context aware application for the exploitation of pervasive environment addressing problems and discussing the important aspects of the selected solution. The requirements of many applications are changing and require the adoption of these technologies. NoSQL databases ensure better performance than RDBMS systems in various use cases, most notably those involving big data. But the choice of the one that best fits the application requirements is a challenge for the programmers that decide to develop a scalable application. There are many differences among the available products and also among the level of maturation on them. From a solution point of view there needs to be a clear analysis of the application context. In particular we focused on technologies that operate in pervasive environments, which can benefit from the huge information available but need to be rethought to extract knowledge and improve the context awareness in order to customize the services. We presented a case study in the field of cultural heritage and its realization using technology standards such as RDBMS systems. After that, we discussed the considerations to be made for upgrading this framework, in particular on the choice of Big Data technologies considering its advantages and disadvantages compared to the effort of re-engineering. Future works will address quantitative analysis of the approach using simulations and benchmarking.

References

1. Gartner: Hype cycle for big data, 2012. Technical report (2012)
2. IBM, Zikopoulos, P., Eaton, C.: Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. 1st edn. McGraw-Hill Osborne Media, New York (2011)
3. Gartner: Pattern-based strategy: Getting value from big data. Technical report (2011)
4. Schroeck, M., Shockley, R., Smart, J., Romero-Morales, D., Tufano, P.: Analytics: The real-world use of big data. IBM Institute for Business Value—executive report, IBM Institute for Business Value (2012)
5. Evans, D.: The internet of things—how the next evolution of the internet is changing everything. Technical report (2011)
6. Stonebraker, M., Cetintemel, U.: One size fits all: an idea whose time has come and gone. In: Proceedings of the 21st International Conference on Data Engineering. ICDE'05, Washington, DC, USA, pp. 2–11. IEEE Computer Society (2005)
7. Gajendran, S.K.: A survey on nosql databases. Technical report (2012)
8. Cattell, R.: Scalable sql and nosql data stores. Technical report (2012)
9. DataStax: A guide to big data workload-management challenges. Technical report (2012)
10. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News **33**, 51–59 (2002)
11. Brewer, E.A.: Towards robust distributed systems (abstract). In: Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, PODC'00, p. 7. ACM, New York (2000)
12. Strauch, C.: Nosql databases (2011) (Online; 26 July 2013)
13. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing STOC'97, pp. 654–663. ACM, New York (1997)
14. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**, 107–113 (2008)
15. Apache: Hadoop (2012) (Online 26 July 2013)
16. Jo Foley, M.: Microsoft drops dryad; puts its big-data bets on hadoop. Technical report (2011)
17. Locatelli, O.: Extending nosql to handle relations in a scalable way models and evaluation framework (2012012)
18. Robinson, I., Webber, J., Eifrem, E.: Graph Databases. O'Reilly Media, Incorporated (2013)
19. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. SIGOPS Oper. Syst. Rev. **41**, 205–220 (2007)

20. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21**, 558–565 (1978)
21. Sumbaly, R., Kreps, J., Gao, L., Feinberg, A., Soman, C., Shah, S.: Serving large-scale batch computed data with project voldemort. (2009)
22. Voldemort: Project voldemort a distributed database. (2012) (Online; 26 July 2013)
23. Memcached: Memcached (2012) (Online; 26 July 2013)
24. Redis: Redis (2012) (Online; 26 July 2013)
25. Riak: Riak (2012) (Online; 26 July 2013)
26. Amazon: Simpledb (2012) (Online; 26 July 2013)
27. Apache: Couchdb (2012) (Online; 26 July 2013)
28. Couchbase: Couchbase (2012) (Online; 26 July 2013)
29. MongoDB: Mongodb (2012) (Online; 26 July 2013)
30. RavenDB: Ravendb (2012) (Online; 26 July 2013)
31. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst. **26**, 4:1–4:26 (2008)
32. HBase: Hbase (2012) (Online; 26 July 2013)
33. Hypertable: Hypertable (2012) (Online; 26 July 2013)
34. Cassandra: Cassandra (2012) (Online; 26 July 2013)
35. BigFoot: Current practices of big data analytics. Technical report (2013)
36. Rabl, T., Gómez-Villamor, S., Sadoghi, M., Muntés-Mulero, V., Jacobsen, H.A., Mankovskii, S.: Solving big data challenges for enterprise application performance management. Proc. VLDB Endow. **5**, 1724–1735 (2012)
37. Neo Technology, I.: Neo4j, the world's leading graph database. (2012) (Online; 26 July 2013)
38. AllegroGraph: Allegrograph (2012) (Online; 26 July 2013)
39. InfiniteGraph: Infinitegraph (2012) (Online; 26 July 2013)
40. findthebest.com: Compare nosql databases (2012) (Online; 26 July 2013)
41. Oracle: Big data for the enterprise. Technical report (2013)
42. Nessi: Nessi white paper on big data. Technical report (2012)
43. Amato, A., Di Martino, B., Venticinque, S.: Bdi intelligent agents for augmented exploitation of pervasive environments. In: WOA, pp. 81–88. (2011)
44. Amato, A., Di Martino, B., Venticinque, S.: Semantically augmented exploitation of pervasive environments by intelligent agents. In: ISPA, pp. 807–814. (2012)
45. Aversa, R., Di Martino, B., Venticinque, S.: Distributed agents network for ubiquitous monitoring and services exploitation. **2**, 197–204 (2009)
46. Renda, G., Gigli, S., Amato, A., Venticinque, S., Martino, B.D., Cappa, F.R.: Mobile devices for the visit of "anfiteatro campano" in santa maria capua vetere. In: EuroMed, pp. 281–290. (2012)
47. Amato, A., Di Martino, B., Venticinque, S.: A semantic framework for delivery of context-aware ubiquitous services in pervasive environments, pp. 412–419. (2012)
48. Amato, A., Di Martino, B., Scialdone, M., Venticinque, S.: Personalized recommendation of semantically annotated media contents. In: Intelligent Distributed Computing VII, vol. 511, pp. 261–270. Springer International Publishing, Switzerland (2013)
49. RDF: Rdf (2012) (Online; 26 July 2013)