Proceedings of International Conference on
"Emerging Research in Computing, Information, Communication and Applications"
ERCICA 2013
ISBN: 9789351071020

A Comparative Analysis of Different NoSQL Databases on Data Model, Query Model and Replication Model

Clarence J. M. Tauro[1,*], Baswanth Rao Patil[2] and K. R. Prashanth[3]

[1]*Christ University, Hosur Road, Bangalore, India.*
[2]*Department of Computer Science, Christ University, Hosur Road, Bangalore, India.*
[3]*Department of Computer Science, Christ University, Hosur Road, Bangalore, India.*
e-mail: clarence.tauro@res.christuniversity.in; baswanth.rao@cs.christuniversity.in; prashanth.r@cs.christuniversity.in

Abstract. Relational databases like SQL have a limitation when it comes to data aggregation, which is used for business intelligence and data mining. Data aggregation becomes impossible on very large volumes of data when it comes to memory and time consumption. NoSQL databases provide an efficient framework to aggregate large volumes of data. There are different NoSQL databases like Key-value stores, Column Family/BigTable clones, Document databases and Graph databases. This paper compares different NoSQL databases against persistence, Replication, Transactions and Implementation language. This paper also discusses on performance and scalability aspects of different NoSQL databases.

Keywords: NoSQL, Key-value stores, BigTable clones, Document databases, Graph databases, Data model, Query model, Replication model.

1. Introduction

Conventional relational database system use two-dimensional table for data creation, with properties like transactions, complex SQL queries, and multi-table related query. However, multi-table queries are not effective for huge data queries. Scalability in relational databases requires powerful servers that are both expensive and difficult to handle.

NoSQL provides the flexibility to store entire data in terms of documents Instead of conventional method of table-row-column. NoSQL is extensively useful when we need to access and analyze huge amounts of unstructured data or data that's stored remotely on multiple virtual servers. There are four different NoSQL databases:

1. Key-value stores: key-value store is a system that stores values indexed for retrieval by keys. These systems can hold structured or unstructured data and can easily be distributed to a cluster or a collection of nodes as in Amazon's DynamoDB and Project Voldemort.
2. Column-oriented databases: Column-oriented databases is a system that stores data in whole column instead of a row, which minimizes disk access compared to a heavily structured table of columns and rows with uniform sized fields for each record as in HBase and Cassandra.
3. Document-based stores: These databases store data and organize them as document collections, instead of structured tables with uniform sized fields for each record. With this database, users can add any number of fields of any length to a document as implemented in Couch DB, Mongo DB.
4. Graph databases: These databases use nodes, edges, and properties to represent and store data in the form of graphs. It is possible to represent data as a graph-like structure, which can be easily traversed as in Allegro Graph and Neo4j.

Comparison among different NoSQL databases can be done against features like:

1. Data Model defines serializers that are responsible to convert byte-arrays into the desired data structures and formats

*Corresponding author

2. Replication in the case of distributed databases means that a data item is stored on more than one node. This is very useful to increase read performance of the database.
3. Consistency means how a system is in a consistent state after the execution of an operation. A distributed system is consistent if after an update operation all readers see their updates in shared data source.

2. Comparison of Different NoSQL Databases

In this section we compare various key-value stores like Amazon Dynamo DB and Project Voldemort

I. Amazon dynamoDB

Amazon DynamoDB is used at Amazon for different purposes. It is influenced by the NoSQL database key-/value-stores; this paper explains in detail about DynamoDB's influencing factors, system design, implementation and applied concepts. DynamoDB at Amazon is used to manage state of services that require high reliability and need tight control over availability, consistency, cost-effectiveness and performance as usage of relational database leads to inefficiencies of limit scale and availability" Voltage Stability [1].

1. Applied concept

DynamoDB uses consistent hashing along with replication as a partitioning scheme. Objects stored in partitions are versioned (multi-version storage) among different nodes. To maintain consistency during updates DynamoDB uses a quorum-like technique and a protocol for decentralized replica synchronization. For the management of members and to detect machine failures it make uses of a gossip-based protocol which helps in addition or removal of servers with a minimal need for manual administration [1].

2. System design

DynamoDB is implemented as a partitioned system with replication and defines with consistency of windows. Therefore, DynamoDB targets applications that operate with weaker consistency for high availability. It does not provide any isolation guarantees [1] that a synchronous replication scheme is not achievable given the context and requirements at Amazon (especially high-availability and scalability). Instead, they have selected an optimistic replication scheme. This features background replication and the possibility for write operations even in the presence of disconnected replicas (due to e.g. network or machine failures). So, as DynamoDB is designed to be "always writeable (i.e. a datastore that is highly available for writes)" conflict resolution has to happen during reads. Since client application has to be equipped with data stores which can perform simple conflict resolutions, for which it has to be aware of such data schema and decide on a conflict resolution method which is best suited for the client's experience and provide such simple scalability DynamoDB feature which addresses both growth in data set size and request rates which allow adding of one storage host at a time [1].

In the DynamoDB, all the nodes have equal responsibilities. There is no node with special roles which can be distinguished. Also, the design favors "decentralized peer-peer technologies over the centralized control" since the latter has resulted in outrages in the past at Amazon. Storage hosts that are added to the system can have heterogeneous hardware that DynamoDB has to consider distributing work proportionally to the capabilities of the individual servers [1].

Table 1. Amazon's dynamoDB – summary of techniques [1].

Problem	Technique	Advantages
Partitioning	Consistent Hashing.	Incremental Scalability.
High priority for writes.	Reads with Vector clocks and reconciliation.	Decoupling of version size from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Failure detection and Membership	Gossip-based membership protocol and failure detection.	Preserves symmetry and prevents centralized registry for storing

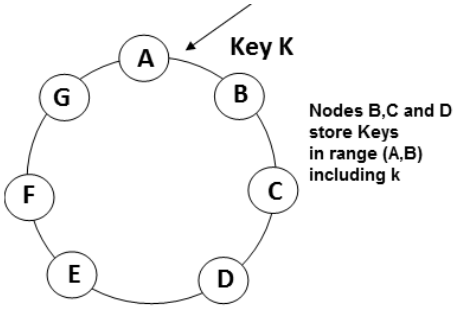

Figure 1. Amazon's DynamoDB – consistent hashing with replication [1].

As DynamoDB is operated in Amazon's own administrative domain, the environment and all nodes are considered non-hostile and therefore no security related features such as authorization and authentication are implemented in DynamoDB [1]. Table 1 summarizes difficulties faced in the design of DynamoDB along with techniques applied to address them and respective advantages.

- System Interface
  DynamoDB provides two operations for client applications:
  - get(key), which returns a list of objects and a context
  - put(key, context, object), with no return value

  The get-operation returns more than one object for a given key if there are any version conflicts, it returns a context for a metadata where object version is stored and clients have to provide this context object in the put operation. The key and object values are handled by DynamoDB as byte-arrays.
- Replication
  To ensure availability and durability in an infrastructure where machine-crashes are the "standard mode of operation" DynamoDB uses replication of data among nodes. Each data item is replicated N-times where N can be configured "per-instance" of DynamoDB [1]. The key k2 with storage node which stores tuple is responsible for replicating versions of tuples from key k to N-1 successors in clockwise direction. There is a list of nodes – called preference list – determined for each key k that has to be stored in DynamoDB. This list consists of more than N nodes as N successive virtual nodes may map to less than N distinct physical nodes.

  Replication is very advantageous that it makes the cluster robust against failures of single nodes. If one machine fails, then there is at least another one with the same data which can replace the lost node. Sometimes it is even useful to replicate the data to different data centres, which makes the database immune against catastrophic events in one area. This is also done to get the data closer to its users, which decreases the latency. But the downsides of data replication are the write operations. A write operation on a replicated database has to be done on each node that is supposed to store the respective data item. A database has basically two choices for doing this: Either a write operation has to be committed to all replication nodes before the database can return an acknowledgment. Or a write operation is first only performed on one or a limited number of nodes and then later sends asynchronously to all the other nodes.
- Implementation and Optimizations
  In addition to the system design, DynamoDB provides suggestions for its optimization [1].

- The code running on a DynamoDB node consists of request coordination, a membership and a failure detection component—each of them implemented in Java.
- DynamoDB provides pluggable persistence components like Berkeley's Database Transactional Data Store and BDB Java Edition [10], MySQL [11], an in-memory buffer with a persistent backing store. In Amazon's DynamoDB applications choose local persistence based on the distribution of object size and the component which processes the coordinating requests is implemented on top of event-driven messaging substrate where message processing pipeline is split into multiple stages. Internodes communication employs Java's New Input Output channels [12].
- When a client issues a read or write request, the contacted node will become its coordinator. Then the requested coordinator receives a stale version of data from a node, it will update it with the latest version. This is called read-repair as it repairs replicas that are missed in recent update.

- To achieve even load-distribution write requests can address to "any of the top N nodes in the preference list".
- As an optimization reducing latency for both, read and write requests, DynamoDB allows client applications to be coordinator of these operations.
- In this case, the state machine created for a request is held locally at a client and periodically contacts a random node and downloads its view on membership. By this clients can determine which nodes are responsible to coordinate for read and write requests in the preference list.
- Alternatively, clients can coordinate write requests locally if the versioning of the DynamoDB instance is based on physical timestamps (and not on vector clocks).
- As write requests usually succeeds read requests, DynamoDB has the capability to further optimize read response by enabling storage node to reply faster to the read coordinator and then is transmitted to the client; in a subsequent write request, the client will contact this node. This optimization enables us to pick the node that was read by the preceding read operation thereby increasing read and write consistency.
- DynamoDB nodes not only serve client requests but also involve in performing a number of background tasks. Admission controller ensures resource division between request processing and background tasks are constantly monitored for the behavior while executing put/get operation. It also monitors disk Input and Output latencies, lock-contention and transaction timeouts which results in failed database access and waiting periods in request queues. By monitoring feedback the admission controller decides the number of time-slices required for the resource access or consumption to be given for the background tasks. It also coordinates the execution of background tasks which have explicit application for resources. [1].

II. Project voldemort

Project Voldemort is a key-/value-store initially developed for and still used at LinkedIn. It provides an API consisting of the following functions: [2].

- get(key), returning a value object
- put(key, value)
- delete(key)

Keys and values are complex compound objects with lists and maps. In Project Voldemort design documentation is discussed compared with relational databases and key-value store which does not provide complex querying capabilities, joins can be implemented in client applications while constraints on foreign-keys, triggers and views are impossible. Nevertheless, a simple concept like the key-/value store offers a number of advantages [2].

- Only efficient queries are allowed.
- The performance of queries can be predicted quite well.
- Data can be easily distributed to a cluster or a collection of nodes.
- In service oriented architectures it is not uncommon to have no foreign key constraints and to do join in the application code as data is retrieved and stored in more than one service or datasource.
- Gaining performance in a relational database often leads to renormalized datastructures or storing more complex objects as BLOBs or XML-documents.
- Application logic and storage can be separated nicely (in contrast to relational databases where application developers might get encouraged to mix business logic with storage operation or to implement business logic in the database as stored procedures to optimize performance).

a) Data format and queries

Project Voldemort allows namespace for key-/value-pairs called "stores" where keys are unique. Each key and values are exactly associated to contain lists and maps as well as scalar values. Operations in Project Voldemort are atomic to exactly one key-/value-pair [4].

b) Versioning and consistency

Like Amazon's DynamoDB Project Voldemort is designed to be highly available for write operations, allows concurrent modifications of data and uses vector clocks to allow casual reasoning about different versions (refer System Design of Amazon DynamoDB). If the datastore itself cannot resolve version conflicts, client applications are

Table 2. Project voldemort – JSON serialization format data types [2].

Type	Storable subtypes	Bytes used	Java-type	JSON example	Definition Example
number	int8, int16, int32, int64, float32, float64, date	8, 16, 32, 64, 32, 64, 32	Byte, Short, Integer, Long Float, Double, Date	1	int32
string	string, bytes	2 + length of string or bytes	String, byte[]	Hello	string
Boolean	Boolean	1	Boolean	True	"Boolean"
object	object	1 + size of contents	Map⟨ String, Object⟩	{"key1":1, "key2":"2", "key3":false}	{"name":"string", "height":"int16"}
array	array	size *sizeof(type)	List⟨?⟩	1, 2, 3	int32

requested for conflict resolution at read time, because it requires little coordination and provides high availability and efficiency as well as failure tolerance. On the downside, client applications have to implement conflict resolution logic that is not necessary in 2PC and Paxos-style consensus protocols [4].

c) Persistence layer and storage engines

Project Voldemort provide pluggable persistency as the lowest layer of the logical architecture allows for different storage engines. To use another existing or self-written storage engine, the operations - get, put and delete instead of an iterator for values will have to be implemented [4].

d) Data model and serialization

Project Voldemort is represented as byte-arrays on keys and values; data models can be configured or a serializer can used to convert byte-array into desired data structures formats for each Voldemort store. Project Voldemort contains pre-defined serializers for the following data structures and formats: JSON (JavaScript Object Notation) is a binary and typed data model which supports the data types list, map, date, Boolean as well as numbers of different precision [5]. JSON can be serialized to and unsterilized from bytes as well as strings. JSON data type helps to communicate with different Project Voldemort instances in a readable format via administration tools like the Voldemort command line client. Java Serialization provided by Java classes where java.io.Serializable interface is implemented. [6,7]. Protocol Buffers are used in Google as they are platform neutral, language neutral, extensible mechanism to serialize structured data with an interface description language to generate code for custom data interchange. They are widely at Google for almost all of its internal RPC protocols and file formats [8,9]. Project Voldemort can be extended by further custom serializes. Correct data interchange is achieved when both data store logic and client applications are aware of each other. It is considered to map well with numerous programming languages as it provides common data types (strings, numbers, lists, maps, and objects) and does not have the object relational mapping problem of impedance mismatch. On the downside, JSON is schema-less internally which causes some issues for applications processing JSON data (e.g. data stores wishing to pursue fundamental checks of JSON-values). As individual schema definition can be defined for each JSON document, this would be ineffective in a datastore of large amount of data which shares the same structure. Project Voldemort therefore offers the possibility to specify data formats for keys and values, as listed in table 2.

3. Comparison Between Document Databases-MongoDB and CouchDB

I. MongoDB

MongoDB is an open-source document-oriented database written in C++ and is completely schema-free and manages JSON-style documents. It focuses on high-performance providing the developer with a set of features to easily model and query data.

a) Data model

MongoDB stores data as BSON objects, which is a binary-encoded serialization of JSON-like documents. It supports all the data types that are part of JSON but also defines new data types, i.e. the Date data type and the BigData type [18] and also support all the data type that is the part of Jason. The key advantage of using BSON is efficiency as it is a binary format [19]. Documents are contained in "collections", they can be seen as an equivalent to relational database tables [17]. Collections can contain any kind of document, no relationship is enforced, and still documents within a collection usually have the same structure as it provides a logical way to organize data. As data within collections is usually contiguous on disk, if collections are smaller better performance is achieved [20]. Each document is identified by a unique ID ("_id" field), which can be given by the user upon document creating or automatically generated by the database [23]. An index is automatically created on the ID field although other indexes can be manually created in order to speed up common queries. Relationships can be modeled in two different ways embedding documents or referencing documents. Embedding documents means that a document might contain other data fields related to the document, i.e. a document modeling a blog post would also contain the post's comments. This option might lead to de-normalization of the database, as the same data might be embedded in different documents. Referencing documents can be seen as the relational database equivalent of using a foreign-key. Instead of embedding the whole data, the document might instead store the ID of the foreign document so that it can fetch. It is important to note that MongoDB does not provide the ability to join documents, therefore when referencing documents, any necessary join has to be done on the client-side [20].

b) Query model

Many traditional SQL queries have a similar counterpart on MongoDB's query Language. Queries are expressed as JSON objects and are sent to MongoDB by the database driver (typically using the "find" method) [13]. More complex queries can be expressed using a Map Reduce operation, and it may be useful for batch processing of data and aggregation operations. The user specifies the map and reduces functions in JavaScript and they are executed on the server side [14]. The results of the operation are stored in a temporary collection which is automatically removed after the client gets the results. It is also possible for the results to be stored in a permanent collection, so that they are always available.

c) Replication model

MongoDB provides Master-Slave replication and Replica sets, where data is asynchronously replicated between servers. In either case only one server is used for write operations at a given time, while read operations can be redirected to slave servers [15]. Replica sets are an extension of the popular Master-Slave replication scheme in order to provide automatic failover and automatic recovery of member nodes [16]. A replicate set is a group of servers where at any point in time there is only one master server, but the set is able to elect a new master if the current one goes down. Data is replicated among all the servers from the set. Figure 2 [15] illustrates the two different replication models.

Since MongoDB has only one single active master at any point in time, strong consistency can be achieved if all the read operations are done on the master [15]. Since replication to the slave servers is done asynchronously and MongoDB does not provide version concurrency control, reads from the slave servers employ eventual consistency semantics. The ability to read from slave servers is usually desired to achieve load balance, therefore the client is also

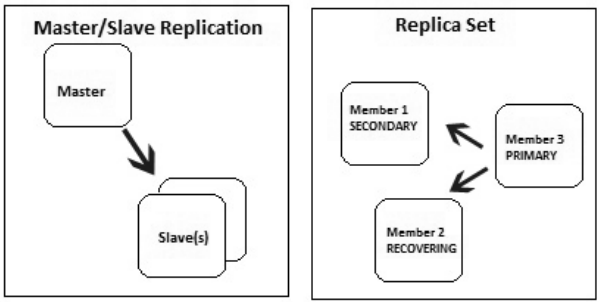

Figure 2. Illustration of master-slave replication and replica sets.

able to enforce that a certain write has replicated to a certain number of slave servers. This feature helps dealing with important writes, where eventual consistency semantics might not be suitable, while at the same time providing the flexibility to read from slave servers.

II. Apache couchDB

Is an open-source document-oriented database and is written in Erlang. It complies with the ACID properties, providing serializability.

a) Data model

Data is stored as semi-structured documents in CouchDB. A document is a JSON file which is a collection of named key-value pairs. Values can be numbers, string, Booleans, lists or dictionaries. Documents are not bound to follow any structure and can be schema-free. Each document CouchDB database is identified by a unique ID (the "_id" field). CouchDB is a simple container of a collection of documents and it does not establish any mandatory relationship between them [22].

b) Query model

CouchDB exposes a RESTful HTTP API to perform basic CRUD operations on all stored items and it uses the HTTP methods POST, GET, PUT and DELETE to do so. More complex queries can be implemented in the form of views (as was seen before) and the result of these views can also be read using the REST API [22].

c) Replication model

CouchDB is a peer-based distributed database system [25] it allows peers to update and change data and then bi-directionally synchronizes the changes. Therefore, we can model either master-slave setups (were synchronizations are unidirectional) or master-master setups were changes can happen in either of the nodes and they must be synchronized in a bidirectional way.

Each document is assigned a revision id and every time a document is updated, the old version is kept and the updated version is given a different revision id. Whenever a conflict is detected, the winning version is saved as the most recent version and the losing version is also saved in the document's history. This is done consistently throughout all the nodes so that the exact same choices are made. The application can then choose to handle the conflict by itself (ignoring one version or merging the changes) [22].

d) Consistency model

CouchDB provides eventual consistency. As multiple masters are allowed, changes need to be propagated to the remaining nodes, and the database does not lock on writes. Therefore, until the changes are propagated from node to node the database remains in an inconsistent state. Still, single master setups (with multiple slaves) are also supported, and in this case strong consistency can be achieved [22].

4. Comparison Between Column-Oriented Databases-HBase and Cassandra

I. HBase

HBase is a free, open-source distributed and column-oriented database. HBase is modelled after Google's Big Table and is written in Java. It is developed as part of Apache Software Foundation's Hadoop project that runs on top of HDFS (Hadoop Distributed File system), which provides BigTable-like capabilities for Hadoop and gives a way to store large amount of data.

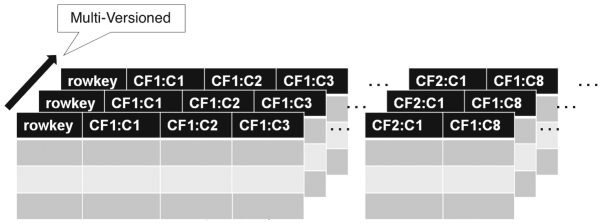

Figure 3. Representation of HBase's data model.

a) Data model

HBase's data model is very similar to that of Cassandra, as they are both based on Google's BigTable. It follows a key-value model where the value is a tuple consisting of a row key, a technologies.

The Column key and a timestamp (i.e. a cell) [28] made of rows and columns, as seen in figure 3. [29]. A row can be a row key with number of columns. The Rows are lexicographically sorted with the lowest order appearing first in the table. The Columns contain a version history of their contents, which is ordered by a timestamp. Columns are grouped into Column Families and they have a common prefix in the form of family: qualifier manner. At column family level both tunings and storage specifications are done, it is advisable that all column family members have the same general access pattern and size characteristics.

b) Query model

HBase provides a Java API which can be used to make queries. The obvious map operations such as put, delete and update are provided by this API [28] An abstraction layer Filter, gives the applications the ability to define several filters for a row key, column families, column key and timestamps and the results can be iterated using the Scanner abstraction. For more complex queries it is possible to write MapReduce jobs which run on the underlying Hadoop infrastructure. Using Pig [31] it is very much possible to query data using a high-level, SQLlike, and language. It is also possible to use Cascading [32] to simplify the process of writing MapReduce jobs. In addition to the Java API, it is also possible to access HBase through REST, Avro or Thrift gateway APIs. A shell for direct user interaction with HBase also exists here.

c) Replication model

HBase uses a simple master-slave replication model [28]. The replication is done asynchronously, that is, the clusters can be geographically distant and the links connecting them can be offline for some time, and rows inserted on the master server may not be available at the same time on the slave servers, hence providing only eventual consistency. Replication is performed by replicating wholeWALEdits in order to maintain atomicity. WALEdits are used in HBase's transaction log (WAL) to represent the collection of edits corresponding to a single transaction.

d) Consistency model

HBase is strictly consistent. Every value appears in one region only which is assigned to one region sever at a time, within the appropriate boundary for its row. If the replication is enabled and one reads it from the slave servers, only eventual consistency can be guaranteed, as the replication is done asynchronously. Row operations are atomic. It is possible to make transactions inside a row. Transactions over multiple rows are unsupported at the moment of this writing [34].

II. Cassandra

Apache Cassandra is a free, open-source distributed, structured key-value store with eventual consistency. It is a top-level project of the Apache Foundation and it was initially developed by Facebook [30]. Cassandra is designed to handle very large amounts of data, while providing high availability and scalability.

a) Data model

A typical instance of Cassandra consists of only one table with a distributed multidimensional map indexed by a key and tables [24] [33]. Rows are generally identified by a string-key and row operations are atomic even if there are many columns being read or written. As in the Bigtable, column-families have to be defined in advance, that is before a cluster of servers comprising a Cassandra instance is launched. The number of column-families per table is infinite however it is expected to specify only few of them. A column family consists of columns and supercolumns12 which can be added dynamically (i.e. at runtime) to column-families and are not restricted in number [33]. Columns have a name and store a number of values per row which are identified by a timestamp (as in Bigtable). Every row in a table has different number of columns. Client specific applications may use ordering of columns within a column family in which name and timestamp can be considered as super-column. Super-columns have a name and number of columns associated with them randomly.

b) Query model

Since Cassandra is in the essence of a key-value store, the querying model provides only three simple operations [31]. A key can access all the values and return the column value associated with that key. Additional methods for getting multiple values or getting multiple columns are also available. But they all rely on the key as input. Additionally it is also possible to run MapReduce jobs on the database, in order to process complex queries. This can be done by using Apache Hadoop [21] which is a framework for distributed data processing. To simplify this task, Cassandra also provides Pig [31] support that enables the user to write complex MapReduce jobs in a high-level language similar to that of SQL. These queries are generally used as an offline method for the analysis of data.

c) Replication model

Since Cassandra is distributed in nature, data is partitioned over a set of nodes and each data item is assigned to a specific node. Data items are assigned to a position by hashing the data item's key, using consistent hashing [17]. Nodes are also assigned a random value within the hashing's space, which represents its position on the ring. To find which node is the coordinator for a given data item, it is necessary to go through the ring in a clockwise manner, and find the first node with a position larger than the data item's position [31]. The client will be able to specify the number of replicas that each data item should have. The data item's coordinator node is then responsible for the replication. Several replication policies are available, were the system might be aware of a node's geographic position (i.e. replicate to other datacentres or to other racks).

5. Comparison Between Graph Databases Allegro and Neo4j

Allegro Graph [35] is one of the precursors in the present generation of graph databases. Although it was born as a graph database, its current development is oriented to meet the Semantic Web standards (viz. RDF/S, SPARQL and OWL). Additionally, AllegroGraph provides some special features for GeoTemporal Reasoning and Social Network Analysis.

Neo4j [26] is based on a network oriented model where relations are first class objects. It implements an object oriented API, and a native disk-based storage manager for graphs, and a framework for graph traversals.

A comparison among databases is typically done by either using a set of common features or by defining a general model used as a comparison basis. The evaluation presented in this section is oriented to evaluate the data model provided by each graph database, in terms of data structures, query language and integrity constraints. As an initial approach let us consider some general features for data storing, operation and manipulation see table 3.

Table 3. Data storing features of graph databases.

Graph database	Main memory	External memory	Backend storage	Indexes
Allegro Graph	✓	✓	✘	✓
Neo4j	✓	✓	✘	✓

Table 4. Operation and manipulation features.

Graph Database	Data definition language	Data manipulation language	Query language	API	GUI
Allegro Graph	✓	✓	✓	✓	✓
Neo4j	✘	✘	✘	✓	✘

Table 5. Graph data structures.

	Allegro graph	Neo4j
Simple graphs	✓	✘
Hyper graphs	✘	✘
Nested graphs	✘	✘
Attributed graphs	✓	✓
Node Labeled	✓	✓
Node Attribution	✘	✓
Directed	✓	✓
Edge Labeled	✓	✓
Edge Attribution	✘	✓

From the point of view of data storing, we review the support for three storing schemas (viz. main memory, external memory, and back-end storage) and the implementation of indexes. It is important to emphasize that managing a huge amount of data is an important requirement in real-life applications for graph databases. Hence the support for external memory storage is a main requirement. Additionally, indexes are the basis to improve data retrieval operations. From the point of view of data operation and manipulation, we evaluate whether a graph database implements database languages, application programming interfaces (API) and graphical user interfaces (GUI). We consider three database languages: the Data Definition Language, that allows modifying the schema of the database by adding, changing, or deleting its objects; the Data Manipulation Language, that allows inserting, deleting and updating data in the database; and the Query Language, which allows retrieving data by using a query expression. Data operation and manipulation features are summarized in table 4.

a) Graph data structures

The data structures refer to the types of entities or objects that can be used to model data. In the case of graph databases, the data structure is naturally defined around the notions of graphs, nodes and edges see table 5.

We consider four graph data structures: simple graphs, hyper graphs, nested graphs and attributed graphs. The basic structure is a very simple flat graph defined as a set of nodes (or vertices) connected by edges (i.e., a binary relation over the set of nodes). A Hyper graph extends this notion by allowing an edge to relate an arbitrary set of nodes (called a hyper edge). A nested graph is a graph whose nodes can be themselves graphs (called hyper nodes). Attributed graphs are graphs where nodes and edges can contain attributes for describing their properties [27]. Additionally, over the above types of graphs, we consider directed or undirected edges, labelled or unlabeled nodes/edges, and attributed nodes/edges (i.e., edges between edges are possible). Note that most graph databases are based on simple graphs or attributed graphs. Only two support hyper graphs and no one nested graphs. We can remark that the hyper-graphs and attributed graphs can be modelled by nested graphs. In contrast, the multilevel nesting provided by nested graphs cannot be modelled by any of the other structures [3].

In comparison with past graph database models, the inclusion of attributes for nodes and edges is a particular feature in current proposals. The introduction of attributes is oriented to improve the speed of retrieval for the data directly related to a given node. This feature shows the influence of implementation issues in the selection and definition of the data structures (and consequently of the data model).

The expressive power for data modeling can be analyzed by comparing the support for representing entities, properties and relations at both instance and schema levels. This evaluation is shown in table 6. At the schema level we found that models support the definition of node, attribute and relation types. Evaluation of several nodes and relations at instance level, an object node is identified by and object-ID represented as instance of a node type. A value node represents an entity identified by a value; a complex node can used to represent a special entity. For example a simple

Table 6. Representation of entities and relations.

	Allegro graph	Neo4j
Node Types	✘	✘
Property Types	✘	✘
Relation Types	✘	✘
Object Nodes	✘	✓
Value Nodes	✓	✓
Complex Nodes	✘	✘
Object relations	✘	✓
Simple relations	✓	✓
Complex relations	✘	✘

relation represents a node instance; special semantics represent a complex relation like grouping, derivation and inheritance. Value nodes and simple relations are supported by all the models. The reason is that both confirm the most basic and simple model for representing graph data. The inclusion of object-oriented concepts (e.g., IDs for objects) for representing entities and relations reflects the use of APIs as the favorite high-level interface for the database. Note that this issue is not new in graph databases. In fact, it was naturally introduced by the so called graph object-oriented data models [2]. Finally, the use of objects (for both nodes and relations) is different of using values. For example, an object node represents an entity identified by an object-ID, but it does not represent the value-name of the entity. In this case, it is necessary to introduce an explicit property or relation "name" in order to specify the name of the entity. The same applies for relations. This issue generates an unnatural form of modeling graph data.

6. Conclusion

The aim of this paper was to give an overview about the different NoSQL Databases, compare them using Data Model, Query Model, Replication Model and consistency Model. NoSQL Databases are effective when we need to process large amount of data with high scalability, to use data among many servers with integration with the programming languages and their data structures compared to the traditional relational database systems. This work also provides knowledge in selecting a NoSQL database and using it effectively. As a result it is expected that it would help adopt NoSQL databases where databases are designed to scale as they grow.

References

[1] DeCandia, Giuseppe, Hastorun, Deniz, Jampani, Madan, Kakulapati, Gunavardhan, Lakshman, Avinash, Pilchin, Alex, Sivasubramanian, Swaminathan, Vosshall, Peter, Vogels, Werner and DynamoDB, Amazon's Highly Available Key-value Store. September (2007). http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-DynamoDB-sosp2007.pdf

[2] Kreps, Jay *et al.*, Project Voldemort – Design. (2010). http://project-voldemort.com/design.php

[3] R. Angles and C. Gutierrez, Survey of graph database models, *ACM Computing Surveys (CSUR)*, vol. 40, no. 1, pp. 1–39 (2008).

[4] http://www.christof-strauch.de/nosqldbs.pdf

[5] D. Crockford, IETF (Internet Engineering Task Force) (Ed.): The application/json Media Type for JavaScript Object Notation (JSON). July 2006. – RFC 4627 (Informational). http://tools.ietf.org/html/rfc4627

[6] Oracle Corporation: Interface Serializable. 1993, 2010.– API documentation of the JavaTM Platform Standard Edition 6. http://download.oracle.com/javase/6/docs/api/java/io/Serializable.html

[7] Bloch, Joshua, Effective Java – Programming Language Guide. Amsterdam: Addison-Wesley Longman (2001).

[8] Google Inc., Google Code – protobuf. (2010). http://code.google.com/p/protobuf/

[9] Google Inc., Google Code – Protocol Buffers. (2010). http://code.google.com/intl/de/apis/protocolbuffers/

[10] Oracle Corporation, Oracle Berkeley DB Products. (2010). http://www.oracle.com/us/products/database/berkeley-db/index.html

[11] Oracle Corporation, MySQL. (2010). http://www.mysql.com/

[12] Oracle Corporation, Java New I/O APIs. (2004,2010). http://download.oracle.com/javase/1.5.0/docs/guide/nio/index.html

[13] Inc. 10gen. SQL to Mongo Mapping Chart – MongoDB (2011). http: //www.mongodb.org/display/DOCS/SQL+to+Mongo+Mapping+Chart,

[14] Inc. 10gen. MapReduce – MongoDB (2011). http://www.mongodb.org/ display/DOCS/MapReduce.
[15] Inc. 10gen. Replication – MongoDB (2011). http://www.mongodb.org/ display/DOCS/Replication.
[16] Inc. 10gen. Replica Sets – MongoDB (2011). http://www.mongodb.org/ display/DOCS/Replica+Sets.
[17] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine and Daniel Lewin, 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, New York, NY, USA (1997) ACM.
[18] BSON. BSON – Binary JSON (2011). http://bsonspec.org/
[19] Inc. 10gen. The MongoDB NoSQL Database Blog – BSON, May (2009). http:// blog.mongodb.org/post/114440717/bson
[20] Inc. 10gen. Schema Design – MongoDB (2011). http://www.mongodb.org/ display/DOCS/Schema+Design
[21] DataStax. Map Reduce | Apache Cassandra Documentation by DataStax (2011). http://www.datastax.com/docs/0.7/map_reduce/index
[22] J. Chris Anderson, Jan Lehnardt and Noah Slater, CouchDB: The Definitive Guide. O'Reilly Media, Inc. (2010).
[23] Inc. 10gen. Object IDs – MongoDB (2011). http://www.mongodb.org/ display/DOCS/Object+IDs
[24] Lakshman, Avinash, Malik and Prashant, Cassandra – A Decentralized Structured Storage System. In: SIGOPS Operating Systems Review, vol. 44, pp. 35–40, April (2010). – Also available online. http://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf
[25] The Apache Software Foundation, Apache CouchDB: Introduction (2011). http: //couchdb.apache.org/docs/intro.html
[26] Neo4j, http://neo4j.org/
[27] H. Ehrig, U. Prange and G. Taentzer, Fundamental theory for typed attributed graph transformation, in *Proc. of the 2nd Int. Conference on Graph Transformation (ICGT)*, ser. LNCS, no. 3256. Springer (2004).
[28] Apache software foundation, The Apache HBase Book (2011). http://hbase.apache.org/book.html
[29] Nguyen Ba Khoa. BigTable Model with Cassandra and HBase (2010). http://aio4s.com/blog/2010/11/08/technology/bigtable-model-cassandra-hbase.html.
[30] The apache software foundation, The Apache Cassandra Project (2011). http://cassandra.apache.org/, last accessed on January (2011).
[31] The apache software foundation, Welcome to Apache Pig! (2011). http://pig.apache.org/
[32] Inc. Concurrent. Cascading (2011). http://www.cascading.org/
[33] Lakshman and Avinash, Cassandra – A structured storage system on a P2P Network. August (2008). – Blog post of 2008-08-25. http://www.facebook.com/note.php?note_id=24413138919
[34] The apache software foundation, HBase ACID Properties (2011). http://hbase.apache.org/acid-semantics.html
[35] AllegroGraph, http://www.franz.com/agraph/allegrograph

