NoSQL database management systems

Raymond Li & Yuhui Pan

Abstract—**The concept NoSQL database has been a hot topic in the age of big data. More and more companies are opting for the cheaper and faster NoSQL solutions for storing and processing large amount of data. In contrast to the traditional RDBMS, many NoSQL implementations sacrificed data integrity and transaction consistency in favor of performance, scalability and flexibility. This paper aims to provide an overview of the NoSQL movement and addresses some of the features and capabilities of different NoSQL data models. Finally, we will discuss some specific implementations methods by investigating three successful commercial NoSQL solutions, Amazon's Dynamo, Apache Cassandra, and the popular MongoDB.**

I. Introduction

1. History and Overview

The development of IBM System R in 1970s has paved the way for many commercially successful relational database management systems still in use today. Since then, relational database management systems (RDBMS) has been dominant force in various industries as it provided sufficient transaction processing speed along with ACID properties and security. There were, however, various attempts of alternative approaches such as XML store and object databases have only been adopted for use by industries that required application for specialized areas.

The 21st century saw the rise of internet giants such as Amazon, Google and Facebook. These companies required high throughput for handling massive amount of online data. These companies thus begin to develop alternative solutions in order to increase performance and scalability. The distaste for big and expensive RDBMS has been the main motivation behind the development of such NoSQL solutions. Many of these projects in-time became open-sourced and were well-appreciated by developers who do not have to worry licensing and commercial supports. As a result, these companies became the driven-force of the modern NoSQL movement and influenced many popular spin-offs that were adopted by various Big Data industries and many web 2.0 startups.

2. Scope

As this is such a general field with tons of explorable topics, it is incredible difficult to summarize everything in 10 pages of paper. This paper will mainly focus on the comparison of NoSQL databases based on data models. Topics including data distribution mechanism, consistency management will only be briefly mentioned in section 4 as part of the discussion for the implementation of specific NoSQL databases. Other advanced topics such as persistency design, scalability features, security, and in-depth comparison between all available database management systems are not covered by this paper.

II. Motivations

1. Unnecessary Complexity

One of the principal driver of the NoSQL movement is the searching for alternatives in areas where relational databases are a bad fit for. Traditional relational databases employ advanced methods for preserving data integrity and maintaining strict consistency for all transactions made to the system. NoSQL databases, on the other hand, utilizes the concept of eventually consistency, meaning that systems will eventually return the last recorded value, but may encounter inconsistent states as upgrades are in progress. This is especially important in productions environments such as large-scale web applications and social media sites, where thousands of transactions are being performed concurrently. The requirement of implementing ACID properties on each individual transaction will greatly hinders performance. For example, it is unnecessary to implement strict consistency for social media comments and likes. Since these data needs to have high availability for read and write, it is unnecessary to maintain ACID properties for each individual transaction when the presentation layer of the application can robustly deal with these inconsistencies.

In many web 2.0 applications, data are stored in simple structures resembling the objects in OOS programming paradigm. In contrast, relational databases maintain a set of expensive object relational mappings that are far better suited for modelling complex real-world entities that are rigidly structured with relations and requires complicated queries. However, for applications data with low complexity, the cost of fitting simple model into object relational mappings is unjustifiable since they can hardly benefit from the features of a RDBMS.

2. Scalibility

The rapid growth of cloud-based service has caused a proliferation in the scale of data being generated and consumed. The increasing demand for scalability has created challenges for many traditional RDBMS.

Relational databases are vertically scalable, to handle increasing loads, hardware upgrades such as CPU, RAM, SSD must be made to the dedicated database server. This is because traditional RDBMS relies on a concept of centralized storage.

When data volume increases, the database must be shared or partitioned by expensive system admins.

In contrast, NoSQL databases are horizontal scalable, meaning that users can just add database servers to the existing NoSQL infrastructure without causing the same operational efforts to perform sharding. This is because most NoSQL databases relies on the idea of distributed storage where workload is shared between different servers, and copies of records are available to all server. This concept is side-effect of giving up features such as the enforcement of constraints and strict consistency in favor for the availability of data. Since NoSQL databases do not requires complex operations that involves the locking across much of the dataset, it is very simple to partition or duplicate the data across multiple servers for processing.

3. Performance

In data-oriented industries, there has always been a demand for higher throughput of data in diverse forms. Relational databases, fitted with rigid schemas, are not well-equipped to deal with unstructured or semi-structured data. NoSQL solutions, however, were created to resolve the storage and processing problem for massive unstructured dataset through its flexible architecture.

Big data handling is also a big issue in relational databases. Since RDBMS doesn't scale well horizontally, the performance tends to drastically decrease as data volume increases, this is especially in the case of unstructured data. The increasing demand for real-time analysis of large evolving data sets added incentives for developing alternate data models for rapid evaluations in contrast to the tradition row storage model. In addition, since most NoSQL data is distributed across multiple servers, parallel computing algorithms including MapReduce can be applied to speed up evaluations. Furthermore, as NoSQL solutions move away from centralized storage, a failure of a single machine within a network has minimum impact with respect to the performance of the entire system. Companies could then save costs by using commodity hardware since failures could easily be dealt with.

4. Flexibility

As mentioned earlier, more flexibility is needed for data that do not fit into a rigid relational model. The current "one size fits all" models utilized by standard RDBMS cannot fulfill the needs for many modern applications. In particular, contents such as images, articles, multimedia can be hard to deal with in a rigid schema. To address this, many NoSQL databases provides document-oriented features to deal with these unstructured data types.

Further, change management is very difficult to deal with in relational databases. Schemas and constraints must be strictly defined before any insertion into the database. Any additional changes to the tables or schema could cause service failure, reduced performance, or additional investments to adapt application modules. In contrast, NoSQL systems can adapt easily to changes. In fact, it gives user the flexibility to store data without advance declaration of schemas. Therefore, it is possible to change the data model without affecting the performance of the system.

III. Classifications

Due to the variety of approaches and the overlapping feature sets developed by NoSQL practitioners to fit their specific requirements, it is difficult to maintain a clear overview of the nonrelational database scene. There have been various approaches to classify NoSQL databases by different criteria and benchmarks, each with different categories and subcategories. In this paper, we will present a classification by data models, which defines the logical organization of data from the perspective client applications, into four major categories.

1. Key-Value

This is one of the simplest data models resembling dictionary or maps and allow clients to insert and request data records by a unique key. This model is usually implement by a Hash Table, where look-ups can be efficient in a extremely large data set. Most databases implementing this concept will keep duplications of records in order to speed up performance and protect against potential failure. This allows Key-Value databases to be capable of handling a very large number of records. They can support high volumes of state changes per second and can handle thousands of concurrent requests through distributed processing and storage. This makes them very useful for storing the results of analytical algorithms as well as real time analysis for Big Data applications.

In contrast to the predefined data types of relational databases, the key-value system treats data as a single opaque collection. This allows greater flexibility in dealing with datasets containing high variation in field types. It allows data storage to more closely conform to modern development principles like the OOP paradigm since optional fields are no longer represented by placeholders and less memory is allocated in dealing with null entries.

However, Key-Value databases are not without its drawbacks. It cannot support data access by values and it's impossible to query by a specific set of values. Solutions that implement clustering can support range selection on keys values, and in some cases, the addition of a secondary index can support multi-indexing of keys. Consistency models varies from different implementations, but most of the times are relaxed in favor of parallel processing performance. The responsibilities of dealing with inconsistencies therefore, shifted towards the client applications.

2. Document-oriented

This data model is very similar to Key-Value store as it also relies on the concept of mapping records using Hash Table like structures. Document-oriented storage tends to be slightly more complex since they encapsulate key-value pairs in document-like structures that can support metadata and clustering. Typically, a primary key is assigned to each document which may be encoded in formats including XML, JSON, and BSON. Similar documents are typically organized into collections for easier accessing. MongoDB is a good example of this type of implementation as we will explore in the next section.

In contrast to the simple Key-Value storage, metadata objects provide a set of API or query language that allows documents-oriented solutions to support the querying of data by the content of the document. The organization of documents as semi-structured model means that there is little separation between data and schema. This allows for greater flexibility as contents that cannot be constrained by schemas can be now be easily represented. In addition, the concept of collections helps the logical organization of documents as well optimizing certain retrieval queries. Depending on the implementation, a document could reside in either one or multiple collections.

3. Column-oriented

Column-oriented storage, sometimes called wide-column store, are databases that store tables by columns instead of rows. This approach has its origin in big data analytics where massive parallel operations are performed on distributed data sets. The general idea is to serialize all the values of one or groups of similar columns in the same location. Many column-oriented solutions support traditional query languages like SQL, but by storing data in columns instead of rows, storage space can be saved as empty fields no longer require addition null values as placeholders. Moreover, since values with similar date types are stored together, it makes data compression and partitioning a lot easier.

For analytical queries that require aggregation of data over a small number of columns, extraction speed can be extremely fast. By architecture of the column storage, the database can precisely access the data instead of scanning and discarding non-relevant values in rows. On hard disk drives, this is particularly important since it exploits the locality of reference and significantly reduce seek times by performing optimal sequential accesses. In contrast to the row-oriented relational databases that spends a lot of resources loading and maintaining indexes, column-based systems have indexes designed to store data, and will only read columns required for certain queries. These characteristics make column-oriented storage optimal for data warehouses, where high scalable solutions are required to perform operations that involves aggregating a vast volume of data.

4. Graph Database

Graph-oriented databases has the most complex data model and utilized graph structures like nodes and edges to represent and store data. They are designed to for datasets whose relations are well-represented as graphs and has elements that are highly connected. They allow simple and fast retrieval of complex hierarchical structures that are difficult to model in traditional databases. Graph databases relies mainly on a schema free model as in order to easily maintain and represent connected data. Records can be modeled as nodes with its relations to other records stored as edges connecting to other nodes.

Complex queries that exploits the connections between entities such as path-finding and searches and be efficiently implemented with graph algorithms. In comparison, to perform a query to analyze the relations between two or more items, a traditional RDBMS would be forced to join multiple tables and getting rid of all the unneeded rows. This is especially important in data structures where parent-child relations are recursively defined, and a simple traversal of the tree can cause significant performance problems for relational databases. In addition, graph databases are generally offers more flexible for storing data entities of various size and type, which allows a more fluidity and realistic data representation at every location.

Data Model	Complexity	Scalability	Performance	Flexibility	Functionality
Key-Value	Minimum	High	High	High	None
Document	Low	High	High	High	Depends
Column	Low	High	High	Moderate	Minimum
Graph	High	Depends	Depends	High	Graph Theory
Relation Database	Moderate	Depends	Depends	Low	Relation Algebra

Figure 1

5. Summary by Data Model

As shown in Figure 1.

IV. Investigation Of Popular NoSQL Solutions

In this section, we will explore four successful commercial NoSQL databases, Amazon's Dynamo, Apache Cassandra, MongoDB and Google Bigtable. As of the writing of this paper, there are more 70 commercially supported NoSQL databases out there. The reason for selecting the four particular solutions are due to their representation for the respectively data model as well as popularity and structural significance. Actually, we would try to plot the big pictures of first three NoSQL database system while diving into Google Bigtable more to illustrate its working mechanism and architecture behind the scenes for the readers to gain a better view of how NoSQL database system functions internally. As mentioned in an earlier section, some advanced features and detailed investigation reports are omitted from this paper.

1. Amazon DynamoDB

Amazon's DynamoDB is one of the pioneers in key-value storage, its design concept influenced a number of NoSQL databases including Riak, Cassandra and Project Voldemort. Its primary usage was the cloud database solution behind Amazon Web Services, an industry-leading cloud computing platform, as well as Amazon's online retailing services. The name was derived and built on top of the concept of Dynamo, a paper published by Amazon to describe a set of techniques to form a highly available, highly decentralized key-value structured storage system, and is what we will be focusing on in this section.

Its motivations came from Amazon's demand for a highly available system with infrastructure made up of thousands of servers located in data centers across the globe. It implements a simple key-value interface while storing the values as BLOBs (Binary Large Objects) for flexibility. Operations are limited to one key-value pair per read/write to avoid cross-referencing other partitions, and only accesses by primary keys are supported. It utilizes the concept eventually consistency and resolves potential conflicts at read time. This allowed key-value pairs to be highly available for writes, making it an ideal tool for services like shopping carts and sessional data management.

In order to make the database highly scalable, the system partitions the data over a set of nodes using the concept of consistent hashing. The idea is to arrange the nodes in a ring by order of the hash value such that the node with the greatest value sits beside the node with the lowest value. If a node is to be inserted or removed from the ring, potential changes will only need to be applied to the neighbouring two nodes. Since the hardware resources associated with each node may not be balanced (servers are not homogeneous), each host is assigned several virtual nodes by taking the hardware capacity of each physical node into account. Dynamo uses the replication of data among nodes to achieve high available and to combat potential machine failures. A coordinator node assigned to each key is responsible for managing the respective replica set, and a vector clock is used to handle data versioning among the nodes.

2. Apache Cassandra

Cassandra was originally developed by Facebook and later released as an open source project under the Apache Software Foundation. It can achieve possibly the highest throughput among all NoSQL solutions while implementing a hybrid data model between key-value and wide-column storage. The highly distributed storage system for managing structured data allows Cassandra to scale to a very large size. It sacrificed consistency by making the data highly available for read and write, making it a popular choice for many high profiled tech giants including Netflix, Reddit, and Apple.

Its data model relies on the concept of column families which are analogous to tables in standard relational databases. In general, a single logical entity can be partitioned into one or more column families depending on the size and variability in the fields. In addition, Cassandra supports the concept of supercolumns such that similar columns can be grouped together and assigned a unique identifier for faster access. By default, only the primary hash index is clustered as it contains the data record in the index entries, although it supports the creation of additional secondary indexes based on the values of any column.

Further, rows within a column family are not clustered, but range scan based on row keys can be supported by the implementation of a user-defined indexing column family. Joining is not supported Cassandra as it is antithetical by nature of the design concept. It can handle a large amount of data in one table and therefore it's best practice to design query-based tables where all requests can be handled in the same partition.

Since disk space is generally the cheapest resource (comparing to CPU, memory, network), Cassandra is designed based on the concept of data duplication. It allows eventually consistency among data replica to achieve high availability, and short response time for data operations. The consistency level and replication policies can be directly defined in the client application, it also features a number of built-in repair features to ensure data remains consistent across replicas.

3. MongoDB

As distributed document-oriented solution, MongoDB is perhaps the most popular choice for web developers due to simple structure and flexible model. From its schema-less design, the storage interface can be entirely defined by the data itself. Its core implementation concept was based on high scalability and low administrative overhead. Its JSON-like storage model helped promote the document-oriented databases family, and is wildly popular solution among web 2.0 startups as it is fast, easy to deploy and featured a set of easy-to-learn JavaScript-based query language.

Mongo's data model utilizes the concept of collections and document for data representation. The documents, stored in BSON, are data structures similar to JSON files but are instead serialized in binary for efficiency. Similar to the column-families in Cassandra, collections are analogous to data tables in RDBMS and contains documents with similar schemas. Each document is assigned a unique identifier as the primary key, and values of the fields can include embedded documents or arrays. Primary key index is not clustered as they contain pointers to the actual documents. By default, MongoDB supports secondary as well as multidimensional indexes which are generally stored as B+ trees for range-based queries.

Due to the distribution of data, MongoDB supports MapReduce operations in JavaScript functions for batch processing of data and aggregation operations. It employs a hash-based distribution and partition the key space of data collections into chunks. Data replications are managed in units of shards, each shard contains a set of processes, sometimes called nodes, that maintain the same data set. The primary node in each shard receives all write requests. After completing the request, the primary node records the transaction in a log, and secondary members apply the action to their respectively managed data set. When a primary node becomes unavailable, a election is held to determine the promotion of a secondary node so the request can be carried out.

4. Google Bigtable

Data Model:

Basically, a Bigtable is a sparse, distributed, persistent multidimensional sorted map that is running on top of the cluster servers. It can use a row key, a column key or a timestamp as its index to map/retrieve data into/from tables. Each value in the table are is an uninterpreted array of bytes of the format:

(row:string, column:string, time:int64) → string

Rows, Columns and Timestamps are the three most important concepts when we deal with data storing and retrieving.

- Rows

The way how Bigtable organize data is by row keys in lexicographic order. A range of row keys would form up another terminology “tablet” in the Bigtable context, which is the unit of distribution and load balancing in the cluster servers. The row keys in a table are arbitrary strings up to 64KB in size. Reading or writing data using row keys is atomically operated as it makes the client easier to reason about the system’s behavior in the presence of concurrent updates to the same row.

- Columns

The syntax of a column key is: family:qualifier. This introduce the concept of Columns Families which is the sets that group data into different categories and also the basic unit of access control. All data stored in a column family is usually of the same type and is compress together if they belong to the same column family. Accessing Bigtable through column keys will yield a small and precise data section. Besides, access control and both disk and memory accounting are performed at the column-family level.

- Timestamps

Timestamps enable Bigtable’s “history” functionality. By using timestamps, each cell would be able to store multiple versions of a single data item. And the enormous flexibility it offers to the application is the definition of user-defined timestamps which means user can insert evolving data into the table by different timestamp and all will stay in the Bigtable. And at the same time in order to keep the cell clean, Garbage Collection comes into play which enable the user to specify the number of versions they would like to keep or discard.

With ingenious design consideration in mind, Google’s Bigtable is able to handle millions of reads/writes per second. And it is one of the most powerful data storage engine in the world.

System Architecture:

BigTable is a distributed hash mechanism built on top of GFS(Google File System). A pool of machines is usually shared between Bigtable cluster and other applications. A central management system is responsible for coordinating jobs scheduling, failure detection and handling as well as resource management and monitoring. As mentioned above, Bigtable maintains data in lexicographic order by row keys. The row range for a table is dynamically partitioned and form up a tablet which is a sequence of 64KB blocks in a data format called SSTable. And SSTable is the fundamental of the GFS, which is so well designed that a data access requires, at most, a single disk access. Make the distribution of data faster and complete. All those establish the precondition that a properly functioning and efficient distributed database system would require.

As could have imagined, such complex system would require coordination from different parts of the system to work together. The main components here are three different types of servers, which are Master Servers, Tablet Servers and Chubby Servers (also known as “Lock Server”).

- Master Server

The job of the master servers is to assign tablets to tablet servers, balances the tablet server load, detects the loss or addition of tablet servers, performs garbage collection and some other chores. It is the core of the Bigtable Service and very lightweight in terms of relieving some of its jobs to the other two types of server.

- Tablet Server

As for the tablet servers, each typically manages between 10 and 1,000 tablets and contains all the data of a group of rows. It is the place where all the reads/writes request traffic for tablets are processed. It is very well backed up by other tablet servers when encountering failures and easily scaled up by splitting tablets into smaller ones.

- Lock Server (Chubby Server)

Chubby servers are the central points holding all the locks for the other two types of servers to avoid collision and ensure consistency. Chubby also rules over tablet server life and death, stores access control lists, data schemas and the bootstrap location of Bigtable data. It communicates with both master server and tablet server to provide information about them as well as assisting synchronization service.

To summarize, a Bigtable cluster stores a number of tables which consists of a set of tablets, and each tablet contains all data associated with a row range. To start off, each table consists of just one tablet. As a table grows, it is automatically split into multiple tablets with the help of tablet server. When reads/writes requests come from the client, the tablet servers handled them and propagate them into GFS. And before all these happens, master and tablet servers have to grab locks from chubby server to proceed and coordinate. Due to the instability of the traffic flow, the tablet servers can be loaded or unloaded dynamically.

Data Replication and distribution:

The underlying GFS is the key element that operate on data replication and distribution. Whereas, file system implementation like GFS is a whole new topic despite what we've discussed above and it is the separate layer that bolster up the database system. We would stop it here in the discussion of GFS since we would mainly focus on the working mechanism and design thinking behind the Bigtable. However, we should still be aware of the import role of GFS in data replication and distribution.

Summary:

It may be difficult to compare between Google Bigtable and RDBMS in terms of working mechanisms since they are so dissimilar and intended goals don't overlap much either. A lot of technical topics implemented by Bigtable like compression, Bloom filters and commit-log are not covered here in order to make our paper more leaning towards big picture side instead of technically detailed side. The benefits we gain from using Google Bigtable is mostly featured by its scalability, its strong ability to process enormous data flow and its tremendous throughput of data traffic. And as a side, it doesn't support joins or SQL type queries. Nonetheless, it does not stop Bigtable from being one of most powerful database system in the world. And those old-style SQL query or join would inevitably fade away in certain areas like the one Bigtable specially fit into. As we can see today, Google Bigtable is quite popular among Google's own products like Google earth and Google Analytics. Admittedly, they are commercially successful software which on the other side endorse our views on this kind of NOSQL database system.

V. CONCLUSION

The previous sections provided a brief overview of the motives and rationales behind the NoSQL movements as well as some common concepts and techniques with respect to some of the core design principles of these solution. Some important topics including security, query optimization, partitioning mechanisms are intentionally left out or slightly mentioned due to generality and complexity nature of the subject matter. The intent of this paper is not to promote NoSQL over RDBMS but to give a conceptual understanding of the benefits to utilize them in certain settings. Weaknesses of various NoSQL databases are uncovered here but they must be carefully evaluated when individuals or companies are selecting a solution in a production environment. Lastly, since NoSQL and is a relatively new area in the ancient field of DBMS, there may still be innovations in technologies that might not have existed as of the writing of this paper.

REFERENCES

[1] C. Strauch, "NoSQL Databases", 2017.

[2] X. Gao, "Investigation and Comparison of Distributed NoSQL Database Systems", Indiana University.

[3] A. Oussous , F. Benjelloun , A. Lahcen , S. Belfkih, "Comparison and Classification of NoSQL Databases for Big Data", Proceedings of International Conference on Big Data, Cloud and Applications, 2015.

[4] G. Matei, "Column-Oriented Databases, an Alternative for Analytical Environment", Database Systems Journal, vol. 1, 2010.

[5] S. Sherif. "Supply cloud-level data scalability with NoSQL databases", IBM Developerworks, 2013.

[6] M. A. Mohamed, O. G. Altrafi, M. O. Ismail, "Relational vs. NoSQL Databases: A Survey", International Journal of Computer and Information Technology, vol. 3 (3), May, 2014.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, "Dynamo: amazon's highly available key-value store", *ACM SIGOPS operating systems review*, vol. 41 (6), 2007.

[8] CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W., WALLACH, D., BURROWS, M., CHANDRA, T., FIKES, A., AND GRUBER, R. Bigtable: A distributed storage system for structured data. In Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06) (2006).

[9] R. P. Padhy, M. R. Patra, and S. C. Satapathy. Rdbms to nosql: Reviewing some next-generation non-relational databases. International Journal of Advanced Engineering Science and Technologies, 11(1):15–30, 2011.