An Overview of Present NoSQL Solutions and Features

Tak-Lon (Stephen) Wu
Computer Science, School of Informatics and Computing
Indiana University, Bloomington, IN
taklwu@indiana.edu

Abstract

NoSQL Database is an emerging research topic as the amount of stored digital information is dramatically growing each minute. In our current era of extreme data scales, NoSQL meets the requirements of the large-scale distributed computing environment, which provides scalability, high availability, high performance and reliability. NoSQL solutions share common features, but feature several different approaches. This study aims to provide an overview of modern NoSQL database solutions and discusses their challenges, features, and use cases.

1. Introduction

The database system is one of the most important components of any kind of large-scale system. Over the last 30 years, SQL database systems, particularly the Relational Database Management System (RDBMS), have dominated this field due to its important ACID [1] model. However, in the recent Cloud Computing age, the amount of stored data grows rapidly to the thousand terabyte and even hundred petabyte scale; thus, the scalability of SQL storage is being questioned. In addition, traditional SQL solutions do not work well with the agile development of modern technologies, which may entail large amounts of complex code and in some cases, even decrease the performance. To address these scalability and performance issues, NoSQL solutions have become a very popular topic.

The phrase NoSQL was first used in 1998 as a name for a lightweight relational database that did not expose the SQL interface. In early 2009, this name was reintroduced by Eric Evans, a Rackspace employee, at an event held to discuss and develop open source distributed databases; he coined the phrase during a conversation with an event organizer. The term NoSQL generally stands for “Not only Structured Query Language,” and the idea of using NoSQL is to fit the data; neither SQL nor the RDBMS model meets the functional requirements. In particular, the continuing trend of cloud computing and the growth of semantic social networks are significant factors that push the development of NoSQL storage in order to provide services for their distributed systems; many Cloud providers have built their own NoSQL solutions such as Google Bigtable [2], Amazon Dynamo [3], Apache HBase [4], and Apache Cassandra [5] to ensure consistency, availability and scalability for their heavy daily workloads and the billions of users they service. However, according to the CAP [6] theorem, there exists tradeoffs in distributed systems among consistency, availability and scalability.

When comparing SQL and NoSQL, discussions normally focus on ACID (Atomicity Consistent Isolation Durability) and BASE (Basically, Available, Soft state, Eventually consistent) [7] models. The ACID model can ensure strict data consistency of online transaction processing while using an exclusive lock mechanism; SQL follows most of these principles in order to provide data identity to applications. It is very important that data must be guaranteed with full consistency even when partitioned, e.g. a commercial account balance is an example of this case. However, according to the CAP theorem, this approach makes SQL data storage difficult to scale with cost and performance issues on a distributed environment when network partitions are required. NoSQL is considered to be solutions and is built beyond the BASE mode to loosen the restrictions with these features. In some senses, large-scale applications that do not require

strong data consistency at every moment benefit from using NoSQL solutions. It is worth mentioning that not all NoSQL solutions operate strictly within the scope of the BASE model. With specific configurations, NoSQL solutions can also function as much as ACID [2-4, 8-10].

This paper is structured as following: Section 2 briefly discusses the challenges for building a NoSQL system. Section 3, we show a classification for existing NoSQL solutions. Section 4, the features and tradeoffs among consistency, scalability, availability, and performance are described. Section 5, we present some use cases. Finally, the paper provides a brief conclusion in Section 6.

2. Design Challenges

Similar to any other large-scale distributed/cloud systems, NoSQL solutions are built with common goals and purposes. They are scalability, availability, and high performance access [11].

Scalability: A fundamental design goal of NoSQL solution is to store unstructured data over a distributed environment, where tables are large and stored separately across nodes. It also aims to provide “unlimited” data capacity for rapidly growing data. Therefore, the design of data model, system architecture and data retrieval model are fundamentally the key factors for supporting these features.

Availability: since data is stored in a distributed fashion, network failures are common when committing updates. So, it must be able to recover from lost data commits and provide up-to-date data access in acceptable range of latency.

High performance access: NoSQL solution is built for content management, data management and computing intensive applications, high latency for any Read/Write operations is not accepted.

3. Classification of NoSQL solutions

More and more organizations and commercial companies have adopted NoSQL solutions to support their projects as data growth has accelerated. Most of these data are unstructured or semi-structured, which does not fit with the traditional SQL database model in which data shares a common table record with standard entities and limited data lengths. Moreover, since unstructured or semi-structured data features different characteristics, different approaches are taken with different types of NoSQL solutions. There are mainly three types of NoSQL database: column-based, key-value based and Documents-based [12].

3.1. Column-based solution

Most of the RDBMS database is organized in tables and for most people, it is easy to understand. Traditional RDBMS stores sets of information in a row-based table. This approach is due to the sequential data access on magnitude hard disk and is suitable for heavy writes record. However, it is not optimized for writing data to a smaller subset of records; in order to update the records, it must read the entire set of tables. In this case, a column-based record solution works well to serve these types of semi-structured data as write-optimized operations. BigTable, HBase, and Cassandra are built based on this feature in order to provide high performance read. All of these solutions provide the ability to have distributed tables across thousands of machines to provide a highly available and scalable storage system. Figure 1 shows a column family structure for this solution.

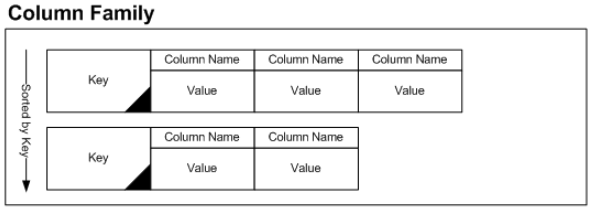

Figure 1. Column-based record

```
{
      "FirstName" : " Sam",
      "LastName" : " Jason",
      "Age" : 60,
       " Title" : " Researcher"
}
```

Figure 2. Document-based record (JSON)

3.2. Key-value-based solution

A key-value based solution stores anything as key-value pairs, which implies stored values retrieved by keys. The key-value pairs can be both structured and unstructured, and it has the advantage of being able to store massive amounts of data, yet retain simple access by a primary key. Generally, the value is an object of implementation language or a string known by the program. Dynamo is the most signature design that uses this model. This system's data is partitioned and replicated using consistent hashing in order to provide scalability and availability. Riak is another example.

3.3. Document-based solution

A Document-based Solution is a type of database that stores uniform fields of each record with non-standard amount of information. As this semi-structured database has no specific schema, information or attributes can be added to any field after it has been inserted into the database; compared to the SQL database, this approach provides flexibility and extendibility. Normally, this type of database uses standard document schema such as XML, JSON, BSON or similar metadata technologies to compose semi-structured information. Here, it must be mentioned that no empty field is allowed to store on this database. Typically, MongoDB [9] and CouchDB [8] are two examples of these technologies. Figure 2 shows a JSON format record of this solution.

4. Features and Tradeoffs

Many NoSQL systems have been built for serving large-scale data distributed over networks, and these systems provide high-level data availability. Table 1 shows the key features of several modern NoSQL solutions, as discussed above. It is interesting to see that all solutions have simple APIs to handle their queries with a small subset of standard SQL-like query language, usually just the get and put functions. This is due to agile data model changes to achieve various goals, which makes higher-level queries less significant to provide general solution to every type of data. Also, NoSQL technologies have become extremely optimized to developers such it is important to clearly understand their needs and purposes based on use cases. For instance, developers could choose either a strong or eventual data consistency model in regard of their system needs. The tradeoffs between these two models, as mentioned above, are availability, durability and performance/throughput.

4.1. Availability

Replication is used to guarantee data availability among these systems. With strong data consistency, all of the above solutions lock or update all the replicas across the distributed environment, which takes a certain amount of time. In other words, before commits are synchronized, data may not be reached. For those applications that are heavily run 24/7, clients cannot accept high latency. One solution is to loose the read access for data with less

persistence concerns, and makes the service becomes highly available in order to serve incoming requests. Thus, the data is being updated in a defined acceptable time frame. Apparently, this may cause inconsistent reading to a data item. For example, Amazon Dynamo can be set to the amount of quorum responses for requests to ensure every available replica has the same set of data, and the waiting time becomes longer as more quorum replicas' confirmations are required.

4.2. Durability and performance

To prevent data loss, records may be flushed to disk before the system returns control to the next operation. The disadvantage of this immediate consistency involves too many disk I/Os and too much latency. Cassandra proposes a per-operation basis consistency to provide flexible write operations without waiting for records to be synchronized to disk. By default, if the Write Ahead Log (WAL) is disable, HBase does not sync log updates immediately to disk; The data are kept in memory, and it will be flush to disk periodically. So, unsaved data in between flush may be lost if any write operations fail. This is due to use cases of HBase, which are primarily to run throughput-oriented batch-processing jobs. CouchDB and MongoDB generally run as an eventually consistent model, where in-memory records are periodically flushed to disk.

5. Use Cases

NoSQL databases generally serve as content management center and store different type of records. In addition, these solutions aim to provide the MapReduce support for large-scale data analytics.

5.1. Content and Data management

ebay [13] presents a use cases for the social signals features using Cassandra to update social-oriented - like/own/wanted functions on item listing pages. eBay has millions of users and listing items, billions of database queries are received every day. In this sense, the social-oriented feeds are huge and must store across data centers. Also, it needs high throughout writes operation to backend databases as the read-insensitive social feeds update massively and frequently. Facebook uses HBase to support their messaging service [14]. The idea is similar - it needs fast-write operations to store record to backend database, it must be scalable due to the huge amount of social feeds, the message body is small, and aims to only retrieve most recently messages. All of these features match with the HBase/HDFS append-only design.

5.2. MapReduce support

Cassandra and HBase are considered as alternative implementations of BigTable, which they natively support Hadoop MapReduce runs on Hadoop Distributed File System (HDFS). [15, 16] study the possibilities of using HBases to stored inverted indices of the original datasets. Indexed records are simultaneously uploaded to HBase by a MapReduce program for data analytics. Other than using Cassandra as a database, DataStax [17] has an enterprise solution which runs it as a HDFS-like file system for serving Hadoop computation. MongoDB and CouchDB are considered as general databases, which do not interface with MapReduce runtimes. Instead, they have their own internal map/reduce implementations for data aggregation and analysis. Dynamo is a completely different database deployment on Amazon cloud; it can communicate with Amazon Elastic MapReduce for importing to and exporting data from HDFS.

6. Conclusion

This paper mainly discusses the NoSQL solutions as large-scale database systems; it does not claim that SQL-type database is insufficient or that it should be replaced by NoSQL-type database. In some cases, such as online bank transactions and commercial sales data warehouses, SQL storage is still better than any current technologies that have been presented in this paper. With different agreement regarding the consistency model, under some circumstances SQL and NoSQL actually function identically to one another.

As the trend of cloud computing and social network continues to impact the way to store data in database, there are certain amounts of pure computer science-related approaches that need the improvements offered by NoSQL technologies to store petabyte scale data. Most of them share the common semi-structured data model, which must aggregate across the complex distributed computing environment. Data and metadata partitions are apparently required in these systems, which provide the ease of scalable solutions to cloud storage. Simultaneously, in order to provide high availability and fault tolerance, those partitions are all duplicated to each machine within the same environment. With either strong or weak consistency, for the general database activities, these solutions provide different mechanisms for satisfying the ACID or BASE model.

In sum, NoSQL is not a brand new database technology; yet, it provides the possibility and flexibility of handling complex semi-structured data and optimizes solutions to different types of data in this massive and data-intensive era of large-scale computing.

NoSQL solution	Data Model	Lang.	Query Model	Sharding	Replication	Consistency	MapReduce Support	Applications
BigTable	Column-based table. Indexs over row-key and columns keys with multiple timestamps	C++	Extendable Application level queries with Google internal C++ client library	Tables are partitioned by row-key into tablets (keys range) stored in different tablet servers. A central metadata server maintains a full overview of the entire system.	Use Google File System (GFS) to store tablets and logs on file level	Strong consistency as each tablet can only store on one tablet server.	Support Google MapReduce	Search engines, high throughput batch data analytics, latency-sensitive database
Cassandra	Column-based table, Indexs over row-key and columns keys with multiple names or timestamps	Java	Extendable Cassandra API initially consist insert, get and delete.	Data are partitioned with an order pre-serving consistent hashing on a distributed "ring" position.	Each data item is replicated as N times depend on system configuration. Use Zookeeper to elect the leader responses for a range of data on the ring.	Eventually consistency, based on the level required by client, the system routes requests to closest or all replicas and wait for quorum response.	Possibly integrated with Hadoop as HDFS-like storage, DataStax's solution currently is the main track	Search engines, log data analytics
HBase	Column-based table. Indexs over row-key and columns keys with multiple timestamps	Java	Shell like command query. Java, REST and Thrift API are supported	Tables are partitioned by row-key into regions stored in different region servers. Similar to BigTable, it has a central metadata server to maintain an overview of all regions.	Use HDFS to store replication with selectable factors	Strong consistency as each record must be updated on assigned region server and replications before read.	Native support for Hadoop MapReduce	Search engines, high throughput batch data analytics
Dynamo	Key-value based data with structured or semi-structured object.	Java	Extendable API initially consist get and put.	Data are partitioned with an order pre-serving consistent hashing on a distributed "ring" position.	Each data item is replicated as N times depend on system configuration. Response nodes handle a range of data on the ring.	Eventually consistency, based on the level required by client, the system routes requests to closest or all replicas and wait for quorum response.	Could be used with Amazon EMR for exporting and importing data from EMR HDFS	Search engines, log data analysis supported by Amazon EMR
CouchDB	Document-based (JSON) with any number of field	Erlang	Provide a HTTP API to views the indexed metadata. Indexed metadata can be created and updated with user-defined MapReduce Functions.	No built-in partitioning, but can be support by the nature of using MapReduce model across different database.	A CouchDB built-in synchronization mechanism using MVCC to replicate data to any instances.	Based on the replication configuration to all replica or a subset replica to provide strong or eventual consistency	MapReduce is supported as internal views functions for data analysis (like, select and where in SQL)	MySQL-like Applications, dynamic queries, less data updates
MongoDB	Document-based (Binary JSON) with structured object	C++	Queries as BSON objects sent to MongoDB driver.	Data are partitioned to shard nodes. Routing services nodes maintain the metadata and handle client requests.	Asynchronous master-slave relationship that one master has the writes and transaction log privilege to a set of replica slaves	Strong consistency when only has one master node, eventual consistency if read send to slave during writes.	MapReduce is supported as internal views functions (like, select and where in SQL)	MySQL-like Applications, dynamic queries, many data updates

Table 1. Modern NoSQL solutions

Reference

[1] Jim Gray and Tony Hey. *The Fourth Paradigm: Data-Intensive Scientific Discovery*, 2010 [accessed 2010 October 21]; Available from: http://research.microsoft.com/en-us/collaboration/fourthparadigm/.

[2] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, *Bigtable: A Distributed Storage System for Structured Data.* ACM Trans. Comput. Syst., 2008. 26(2): p. 1-26. DOI:10.1145/1365815.1365816

[3] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels, *Dynamo: amazon's highly available key-value store.* SIGOPS Oper. Syst. Rev., 2007. 41(6): p. 205-220. DOI:10.1145/1323293.1294281

[4] Apache. *Hbase implementation of Bigtable on Hadoop File System*, 2010 [accessed 2010 June 5]; Available from: http://hbase.apache.org/.

[5] Avinash Lakshman and Prashant Malik, *Cassandra: a decentralized structured storage system.* SIGOPS Oper. Syst. Rev., 2010. 44(2): p. 35-40. DOI:10.1145/1773912.1773922

[6] Seth Gilbert and Nancy Lynch, *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.* SIGACT News, 2002. 33(2): p. 51-59. DOI:10.1145/564585.564601

[7] Dan Pritchett, *BASE: An Acid Alternative.* Queue, 2008. 6(3): p. 48-55. DOI:10.1145/1394127.1394128

[8] J. Chris Anderson, Jan Lehnardt, and Noah Slater, *CouchDB: The Definitive Guide Time to Relax*. 2010, ISBN: 0596155891, 9780596155896: O'Reilly Media, Inc. p.272.

[9] Eelco Plugge, Tim Hawkins, and Peter Membrey, *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing*. 2010, ISBN: 1430230517, 9781430230519: Apress. p.328.

[10] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni, *PNUTS: Yahoo!'s hosted data serving platform.* Proc. VLDB Endow., 2008. 1(2): p. 1277-1288.

[11] B. G. Tudorica and C. Bucur. *A comparison between several NoSQL databases with comments and notes*. in *Roedunet International Conference (RoEduNet), 2011 10th*. 2011.

[12] N. Leavitt, *Will NoSQL Databases Live Up to Their Promise?* Computer, 2010. 43(2): p. 12-14. DOI:10.1109/mc.2010.58

[13] Jay Patel. *Buy It Now! Cassandra at eBay*, 2012; Available from: http://www.datastax.com/wp-content/uploads/2012/08/C2012-BuyItNow-JayPatel.pdf.

[14] Dhruba Borthakur, Joydeep SenSarma, and Jonathan Gray, *Apache Hadoop Goes Realtime at Facebook*, in *SIGMOD*. 2011, ACM: Athens, Greece. p. 4503-0661.

[15] Xiaoming Gao, Vaibhav Nachankar, and Judy Qiu, *Experimenting lucene index on HBase in an HPC environment*, in *Proceedings of the first annual workshop on High performance computing meets databases*. 2011, ACM: Seattle, Washington, USA. p. 25-28.

[16] Ioannis Konstantinou, Evangelos Angelou, Dimitrios Tsoumakos, and Nectarios Koziris, *Distributed indexing of web scale datasets for the cloud*, in *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*. 2010, ACM: Raleigh, North Carolina. p. 1-6.

[17] *DataStax*; Available from: http://www.datastax.com/.