NoSQL in Cloud Computing

Ruisheng Fu	Guanyu Guo	Chen Sui
21421190	21421189	21421183
CCNT Lab.	LIST Lab.	

January 3, 2015

Contents

Abstract

A lot of changes in database management system has been made since the inception of cloud computing. Such critical and increasing needs within cloud computing as scalability, elasticity and processing a huge amount of data can be fulfilled by the NoSQL databases as opposed to RDBMS[1]. In this report we have dived into several primary NoSQL databases used in leading cloud vendors, summarizing and discussing detailed techniques as well as analysis with comparisons.

[1]Relational Database Management System

1 Introduction

Cloud computing has been a evolving computing terminology that is very much in the public eye. It is its responsibility to manage and group remote servers that allow data storage and online access to various services.

In the field of computing, the various advancements and aspects are key evidences that explain the reason why higher priorities are given to scalability, resource utilization and power savings, with respect to data storage, rather than consistency. The traditional RDBMSs offer functionalities like clustering, synchronization (always consistent), load balancing and structured querying. However, what classical RDBMS could not do so well is to scale[2] to heavy workloads compared to NoSQL databases. As non-relational databases have cropped up both inside and outside the cloud, there comes heated debate around SQL and NoSQL.[1]

The two solutions, manual sharding and caching, applied to classical SQL databases are not adequate enough to cope up with the modern web applications, thus agility can't be achieved. On the contrary, NoSQL databases is designed to handle such sort of problems. In those applications where high availability, speed, fault tolerance or consistency are needed, NoSQL is the choice, in that it is designed to scale out, to provide elasticity and to be highly available. The misleading term *NoSQL* should be seen as the definition[10] that is "Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable", and is mostly translated with "Not only SQL".

In this report, we firstly examine several major NoSQL databases implemented and used in cloud vendors like Google, Amazon, and Yahoo, along with description about the main ideas of each design. Afterwards, analysis towards different NoSQL databases and we present benchmarking of top NoSQL as a visualizing comparison. Finally, a brief summarization is included in conclusion and further studies as well as challenges is discussed.

2 Major Streams

Currently there are approximately 150 NoSQL databases categorized by data models into a bunch of classes.[10] We review certain amount of prevailing

[2]Scaling, in a google sense, means that an application runs on small commodity PC hardware, but supports essentially unbounded load as more PC's are added.

NoSQL databases through several aspects, including data model, partitions, availability, and consistency. [9] makes a brief introduction to MongoDB with an installation guide, and mainly focus on the performance benchmark.

2.1 Data Model

The major data models adopted we are going to investigate are Wide Column Store (Column Families), Document Store, and Key Value Store, whereas the minor ones (Graph Databases, Object Databases, etc.) are beyond our scope of discussion in this report.

Column Family Store Columnar databases are logically similar to tabular databases. The difference is that the data are column-wise stored and retrieved.

Bigtable[2] is a sparse, distributed, persistent multidimensional sorted map introduced by Google. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

$$(row : string, column : string, time : int64) \rightarrow string$$

Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings.

The row keys in a table are arbitrary strings. Bigtable maintains data in lexicographic order by row key. The row range for a table is dynamically partitioned. Each row range is called a tablet, which is the unit of distribution and load balancing.Column keys are grouped into sets called column families, which form the basic unit of access control. A column family must be created before data can be stored under any column key in that family; after a family has been created, any column key within the family can be used. A column key is named using the following syntax: family:qualifier.Each cell in a Bigtable can contain multiple versions of the same data; these versions are indexed by timestamp.

Casandra comes under column family. It is designed to process the data which are spread across different servers without a single point failure. Columns in Cassandra are grouped together very much similar to what

happens in the Bigtable system. Cassandra[6] exposes two kinds of columns families, *Simple* and *Super* column families. Super column families can be visualized as a column family within a column family. Casandra also allows columns to be sorted either by time or by name, which is often exploited by different applications.

Document Store A document-oriented database eschews the table-based relational database structure. MongoDB is the well-known member of the family. In general, it stores business subjects in the minimal number of documents instead of breaking it up into relational structures[3] in favor of JSON-like formats with dynamic schemas.[9] This flexibility facilitates the mapping of documents to an entity or an object in MongoDB, in which there are two tools to allow applications to represent relationships between data: *references* and *embedded documents*.[7]

Key Value Store

2.2 Availability

As server downtime implies lost revenue, high availability is the key factor to sustain services. To achieve this goal, various methods has emerged.

Replication Replication is one way to ensure consistency between redundant resources, to improve reliability, fault-tolerance, or accessibility.

There are two types of replication supported in MongoDB: *master-slave* and *replica sets*.[9] The latter works the same as the former, except that it is possible to elect a new master if the original master went down.

Unlike HBase, Cassandra uses a coordinator node in charge of the replication of the data items and locally stores each key within its range. It also provides three replication policies: *Rack Unaware*, *Rack Aware*, and *Datacenter Aware*. For "Rack Unaware" replication strategy, the con-coordinator replicas are chosen by picking certain amount of successors[3] of the coordinator on the ring. The rest two strategies involve a leader node elected by Zookeeper.[4]

[3]Rigorously, $N-1$ replicas are picked where N is the replication factor.

Failure Detection Failure detection is a mechanism by which a node can locally determine if any other node in the system is up or down. It is vital to support cluster membership.

In Cassandra, a modified version of the Φ Accrual Failure Detector is used. The main idea of the detector is to emit a value representing a suspicion level, rather than a boolean value, for each of monitored nodes. All that the value Φ conveys is the likelihood that we will make a mistake. [6] also claims that AFD is good in both accuracy and speed and adjust well to network or load conditions.

2.3 Partition

Due to huge amount of data across applications, it needs to partition the data to distribute it over a cluster. There are three main approaches for partitioning: *sharding*, *range partition* and *hash partition*. The ability to dynamically partition the data over nodes ensures scaling incrementally.

Sharding Sharding is a method for storing data across multiple machines. (http://docs.mongodb.org/manual/core/sharding-introduction/)

Another feature supported by MongoDB is automatic sharding [7]. Using this feature the data can be partitioned over multiple nodes. The administrator only has to define a sharding key for each collection which defines how to partition the contained documents. In such an environment, the clients connect to a special master node called mongos process which analyses the query and redirects it to the appropriate node or nodes. To avoid data losses, every logical node can consist of multiple physical servers which act as a replica set. Using this node infra- structure it is also possible to use Map/Reduce [8] to work on the available data set having a very good performance.

Range What range partition does is to order records lexicographically based on keys and divide it according to the ordering result.

Hashing By hashing, we mean that hash records based on key to a linear space and then divide space among different servers.

Cassandra partitions data across the cluster using *consistent hashing* but uses an order preserving hash function to do so. In [6], it points out the challenges (non-uniform data and load distribution) the basic consistent hashing

algorithm[5] is facing, and adopt the latter of two suggested ways in [8] to address these issues, because it makes the design and implementation tractable and helps to make choices about load balancing.

2.4 Consistency

The Cassandra system relies on the local file system for data persistence. Typical write operation involves a write into a commit log for durability and recoverability and an update into an in-memory data structure. The write into the in-memory data structure is performed only after a successful write into the commit log.

Over time many such files could exist on disk and a merge process runs in the background to collate the dif- ferent files into one file. This process is very similar to the compaction process that happens in the ****Bigtable system****. pro We have a dedicated disk on each machine for the commit log since all writes into the commit log are sequential and so we can maximize disk throughput.

3 Performance

Cassandra:

There are two kinds of search fea- tures that are enabled today (a) term search (b) interactions - given the name of a person return all messages that the user might have ever sent or received from that person.

In order to make the searches fast Cassandra provides certain hooks for intelligent caching of data.

result very good

4 Conclusion

CAP consistency, availability, partition only 2 of 3

comment: It really depends on the needs of your application. For web applications that have light querying, key/value stores are very useful. For enterprise databases where reporting is typically very heavy, relational databases fit better.

cassandra:

We have built, implemented, and operated a storage sys- tem providing scalability, high performance, and wide appli- cability. We have empirically demonstrated that Cassandra can support a very high update throughput while deliver- ing low latency. Future works involves adding compression, ability to support atomicity across keys and secondary index support.

4.1 Drawback

Transactions are not directly supported by MongoDB. Though there are two workarounds: atomic operations and two-phase commits. Atomic operations allow performing multiple operations in one call. An example is findAndModify [9] or the inc [10] operator used in updates. There are several other limitations. For example, if you use the 32-bit version of MongoDB the data set is limited to a size of 2.5 gigabytes [11]. MongoDB does not support full single server durability which means you need multiple replications to avoid data losses if one server suffers a power loss or crash [12]. Another drawback is the fact that it uses much more storage space for the same data then for example PostgreSQL. Because – as opposed to relational databases – every document can have different keys [13] the whole document has to be stored, not only the values. That's why it is recommended to use short key names.

References

[1] Tony Bain. Is the Relational Database Doomed?, 2009. URL: http://readwrite.com/2009/02/12/is-the-relational-database-doomed.

[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. In *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, pages 205–218, 2006. URL: http://research.google.com/archive/bigtable-osdi06.pdf, doi:10.1145/1365815.1365816.

[3] Steve Hoberman. *Data Modeling for MongoDB*. Technics Publ., 2014.

[4] Patrick Hunt, Mahadev Konar, FP Junqueira, and Benjamin Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems.

USENIX Annual Technical ..., 8:11–11, 2010. URL: http://portal.acm.org/citation.cfm?id=1855851$\delimiter"026E30F$nhttps://www.usenix.org/event/usenix10/tech/full_papers/Hunt.pdf.

[5] David Karger, Tom Leightonl, Daniel Lewinl, 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 *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, 1997. doi:doi:10.1145/258533.258660.

[6] Laksham Avinash and Prashant Malik. Cassandra: a decentralized structured storage system, 2010. URL: http://dl.acm.org/citation.cfm?id=1773922, doi:10.1145/1773912.1773922.

[7] MongoDB Inc. Data Modeling Introduction, 2009. URL: http://docs.mongodb.org/manual/core/data-modeling-introduction/.

[8] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11:17–32, 2003. doi:10.1109/TNET.2002.808407.

[9] Rico Suter. MongoDB: An Introduction and performance Analysis. In *Seminar Thesis, Rapperswil*, 2012.

[10] Unknown. NOSQL Databases, 2012. URL: http://nosql-database.org/.