NoSQL in Cloud Computing

Ruisheng Fu
21421190
CCNT Lab.

Guanyu Guo
21421189
LIST Lab.

Chen Sui
21421183

January 4, 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[13] 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.[13] 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.

Bigtable[2] designed by Google describes the well-known data model, which gives clients dynamic control over data layout and format and has successfully provided a flexible, high-performance solution for many Google products. The Amazon Dynamo[3] presents a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. Cassandra[7] concludes design model and implementation part and claims that it is designed to achieve high write throughput and without sacrificing read efficiency, which is a combination of Bigtable and Dynamo. [10] describe the motivation for PNUTS and its table storage and replication layers, and finally performance analysis with experimental results. Paper [8] describes the architecture and implementation of Dremel, and explains how it complements Map/Reduce-based computing. We present a novel columnar storage representation for nested records and discuss experiments on few-thousand node instances of the system. [12] 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 kind of sparse, distributed, persistent sorted map introduced by Google. It is indexed by three elements: row key, column key, and timestamp like this:

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

Each value in this map is an uninterpreted array of bytes. The row keys which is used to maintains data in lexicographic order are arbitrary strings. Column key consist of family and qualifier. Column keys are grouped into sets called column families, which form the basic unit of access control. Besides, Bigtable can contain multiple versions of each cell by indexed timestamp.

Similarly, Casandra comes under column family. Columns in Cassandra are grouped together very much similar to what happens in the Bigtable system. Cassandra[7] 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[4] in favor of JSON-like formats with dynamic schemas.[12] 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*.[9]

Key Value Store Dynamo is Amazon's key-value storage system that procides an "always-on" experience. Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performace. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios.

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*.[12] The latter works the same as the former, except that it is possible to elect a new master if the original master went down.

PNUTS does not rely on log or archive data. Instead, it use a guaranteed delivery pub/sub mechanism to act as a redo log, replaying updates which are lost before being applied to disk due to failure. It replicate the data to multiple regions providing additional reliability, obviating the need for archiving or backups.

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.[5]

Replications of Dynamo[3] resembles Cassandra. It replicates its data on multiple hosts, Each data item is replicated at N hosts using the same consistent hashing.

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 Dynamo, using hinted handoff ensure that read and write operations are not failed due to temporary node or network failtures. Applications that need the highest level of availability can set W to 1, which ensures that a write is accepted as long as a single node in the system has durably written the key it to its local store.

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. [7] also claims that AFD is good in both accuracy and speed and adjust well to network or load conditions.

Bigtable relies on a highly-available and persistent distributed lock service called Chubby. Bigtable uses Chubby for a variety of tasks: to ensure that there is at most one active master at any time; to store the bootstrap location of Bigtable data; to discover tablet servers and finalize tablet server deaths; to store Bigtable schema information (the column family information for each table); and to store access control lists.

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

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 two main approaches for partitioning: *range partition* and *hash partition*. The ability to dynamically partition the data over nodes ensures scaling incrementally.

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

Besides the column family described in section 2.1, 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. As a result, reads of short row ranges are efficient and typically require communication with only a small number of machines. In this manner, clients can exploit this property by selecting their row keys so that they get good locality for their data accesses.

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

Dynamo's partitioning scheme relies on consistent hashing[4] to distribute the load across multiple storage hosts.

Cassandra also partitions data across the cluster using *consistent hashing* but uses an order preserving hash function to do so. In [7], it points out the challenges (non-uniform data and load distribution) the basic consistent hashing algorithm[6] is facing, and adopt the latter of two suggested ways in [11] to address these issues, because it makes the design and implementation tractable and helps to make choices about load balancing.

Combining these two, PNUTS allows applications to declare tables to be hashed or ordered[5], supporting both workloads efficiently.

Sharding In addition to two ways above, we also like to mention *sharding*. Sharding is a method for storing data across multiple machines.

Automatic sharding is one feature supported by MongoDB. The administrator only has to define a sharding key for each collection. In such an

[4] In consistent hashing, the output range of a hash function is treated as a fixed circular space or "ring". Each node in the system is assigned a random value within this space which represents its "position" on the ring.

[5] Hashing is fully supported while order range is claimed as their future work.

environment, the clients connect to a special master node which analyses the query and redirects it to the appropriate node(s). To avoid data losses, every logical node can consist of multiple physical servers which act as a replica set. Thus it is also possible to use Map/Reduce to work on the available data set having a very good performance.

2.4 Consistency

In Dynamo, the consistency is maintained by a quorum-like technique and a decentralized replica synchronization protocol. Dynamo provides eventual consistency, which allows for updates to be propagated to all replicas asynchronously.

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. Since Cassandra borrows from Bigtable, the process which runs in the background to collate such files into one 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.

PNUTS provides a consistency model that is between the two extremes of general serializability and eventual consistency. Our model stems from our earlier observation that web applications typically manipulate one record at a time, while different records may have activity with different geographic locality. We provide per-record timeline consistency: all replicas of a given record apply all updates to the record in the same order. An example sequence of updates to a record is shown in this diagram: This model is implemented as follows. One of the replicas is designated as the master, independently for each record, and all updates to that record are forwarded to the master. The master replica for a record is adaptively changed to suit the workload – the replica receiving the majority of write requests for a particular record becomes the master for that record. The record carries a sequence number that is incremented on every write. As shown in the diagram, the sequence number consists of the generation of the record (each new insert is a new generation) and the version of the record (each update of an existing record creates a new version). Note that we (currently) keep only one version of a record at each replica.

3 Performance

Cassandra:

There are two kinds of search features 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

Bigtable has set up a Bigtable cluster with N tablet servers to measure the performance and scalability of Bigtable as N is varied. N client machines generated the Bigtable load used for these tests. The result is presented on fig.

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: It is designed to process the data which are spread across different servers without a single point failure.

We have built, implemented, and operated a storage system providing scalability, high performance, and wide applicability. We have empirically demonstrated that Cassandra can support a very high update throughput while delivering low latency. Future works involves adding compression, ability to support atomicity across keys and secondary index support.

Recent interest in massively scalable databases has produced several systems, each optimized for a different point in the design space. Google's BigTable [8] provides record oriented access to very large tables, but to our knowledge there have been no publications describing support for geographic replication, secondary indexes, materialized views, the ability to create multiple tables, and hash-organized tables. Amazon's Dynamo [12] is a highly-available system that provides geographic replication via a gossip mechanism, but its eventual consistency model does not adequately support many applications, and it does not support ordered tables. Other large scale distributed storage systems include Amazon's S3 and SimpleDB services, and Microsoft's CloudDB initiative, but there is little information publicly available about the architecture of these systems.

Both PNUTS and Dynamo are distributed data storage systems and are used as scalable back-ends for the various online applications and services of Yahoo and Amazon respectively. This study reveals that both of these systems have some common features and design considerations and are yet distinct with regards to their architectures and implementations.

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. There are several other limitations. 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. 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 the whole document has to be stored, not only the values. That's why it is recommended to use short key names.

1 Indexes and Materialized Views In order to support efficient query processing, it is often critical to provide secondary indexes and materialized views. An index/view maintainer will listen to the stream of updates from message broker, and generate corresponding updates. For example, if a user moves from Wisconsin to California, and we have an index on location, the maintainer will delete the Wisconsin index entry for the user and insert a California index entry for the user. Further research is needed to examine the semantic implications of answering queries using possibly stale indexes and views. 2 Bundled Updates Several customers have expressed a need for an extension of the consistency guarantees we provide. The extension, called bundled updates, provides atomic, no isolated updates to multiple records. That is, all updates in the bundle are guaranteed to eventually complete, but other transactions may see intermediate states resulting from a subset of the updates. For example, if Alice and Bob accept a bi-directed social

network connection, we need to update both Alice's and Bob's records to point to the other user. Both updates need to complete (and the application writer would prefer not to check and retry to ensure this, as in the current system) but it is not critical to provide serializability; it is ok if Alice is temporarily a friend to Bob but not vice-versa. The challenges in implementing bundled updates are to ensure the timeline consistency guarantees described in Section 2.2 when the updates in the bundle are asynchronously and independently applied, and to provide a convenient mechanism for the client to determine when all updates in the bundle have completed. 3 Batch-Query Processing Although PNUTS is optimized for web OLTP workloads, we believe that it can also serve as a data store for batch and bulk processing, such as that provided by MapReduce or Pig. This requires further investigation of how a scan-oriented bulk workload interacts with a seek-oriented serving workload. It may be necessary to separate PNUTS replicas into "batch" and "serving," and optimize them separately for the different workloads. Also, parallel batch systems optimize their execution based on the current location of data, and therefore we may need to provide hooks for accessing tablets directly, bypassing routers.

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] 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. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, volume 41, pages 205–220,

2007. URL: http://portal.acm.org/citation.cfm?doid=1323293.1294281, doi:10.1145/1323293.1294281.

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

[5] 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.

[6] 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.

[7] 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.

[8] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive Analysis of Web-Scale Datasets. In *36th International Conference on Very Large Data Bases*, pages 330–339, 2010. URL: http://www.vldb2010.org/accept.htm, doi:10.1145/1953122.1953148.

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

[10] Adam Silberstein, Brian F. Cooper, Utkarsh Srivastava, Erik Vee, Ramana Yerneni, and Raghu Ramakrishnan. PNUTS: Yahoo!'s Hosted Data Serving PLatform. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, page 765, 2008. URL: http://portal.acm.org/citation.cfm?doid=1376616.1376693, doi:10.1145/1376616.1376693.

[11] 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.

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

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