An Empirical Evaluation of Distributed Key/Value Storage Systems

Group #8
Yi Qu
Shivam Kulkarni
Yunan Zhang

CS 550 - Advanced Operating Systems (Fall 2019)
Prof. Ioan Raicu

Abstract:

Distributed Storage Systems are the most popular storage manage schemes nowadays, especially with the emergence of big data and cloud computing technologies. The traditional RDBMS technology shows bottleneck when horizontal scalability is needed, which means, in order to improve the performance of RDBMS, the system administrators need to update the hardware, which creates the upper bound of scalability. NoSQL techniques make it possible to scale the distributed storage system without encountering much reduction in terms of latency and throughput. In this paper, we compared four popular distributed key/value storage systems, namely MongoDB, Cassandra, Redis, and ZHT. We conducted scalable experiments and measured their performance with respect to latency and throughput.

Keywords: *Distributed Hash Table, Distributed Storage Systems, Key/value pairs*.

1. Introduction

NoSQL databases promise faster and efficient performance than the legacy RDBMS[1] . In RDBMS, data are stored in tabular format, which requires homogeneous schema. Whereas, NoSQL database provides a more flexible schema. Data can be stored as either key/value pairs or json document forms. In consequence, data entries in RDBMS must have the same attributes, but NoSQL database allows flexible attributes for each entry. The main difference between NoSQL and traditional RDBMS is, the NoSQL model is designed for processing huge amount of data in a second, with relatively low consistency requirements[2]. On the contrary, RDBMS, limited by the ACID[1] restriction, sacrifices the scalability performance, which is replaced by BASE[2] constraint in NoSQL database. ACID restrictions are mainly used in transactions since it ensures atomic actions. However, BASE ensures its reliability in spite of loss of Consistency[3].

The scalability is the most important issue when comparing and selecting among many NoSQL databases to fit different implementation scenarios. There are many effective methods to measure the scalability of distributed storage systems. In this article, we select throughput and latency, since throughput measures how many operations can be done in the unit amount of time, and latency measures how much time it takes to finish one operation, both throughput guarantee and latency guarantee are equally critical to measure the performance of large scale systems[4].

As for the format of data, we choose key/value pairs. NoSQL database supports many different data formats to be stored in the entry. Among them, key/value pair is easy to implement and control. We can specify the size of key and value in order to measure the performance of operations regardless of the difference in the size of data. In the experiment section, we require the key size to be 10 bytes and value size to be 90 bytes.

[1] ACID stands for Atomic, Consistency-preservation, Isolation, and Durability.
[2] BASE stands for Basically Available, Stable state, Eventually consistent.

2. Background

2.1 Cassandra:

Cassandra, one of the biggest projects by Apache, is an open-source, wide-column store based database management system, which is capable of handling large amounts of data. It has certainly made its name in the Big-Data world as it earned the reputation of a highly available system with no single point of failure. Cassandra is designed for large scale projects. As the cluster grows, the probability of node failure increases. To ensure scalability and reliability, Cassandra manages to maintain a persistent state. To tackle the issue of node failures, Cassandra employs a peer-to-peer distributed system across homogeneous nodes where data is distributed among all nodes in the cluster. Cassandra introduces an in-memory structure called memtable. Data is indexed and written to a memtable. To ensure durability, every node's activities are captured through commit logs. When memtable content exceeds the configurable threshold or the commitlog exceeds the commit log space, the contents of the memtable are flushed flushed to disk in an SSTable data file. Cassandra is schema-less, and all of the data consist of an indexed key and a value. Cassandra uses a log-structured merge tree as its storage structure. Cassandra avoids reading before writing. This may lead to corrupted caches and increased I/O requirements. To avoid this, the storage engine groups the inserts/updates to be made and then append only the updated parts of a row. Cassandra never re-reads/ re-writes and never overwrites existing data. Cassandra creates sub-directories for the tables within each keyspace directory. This allows to symlink (symbolically link) a table into a physical driver. This also provides the capability to move a table to faster media like SSD’s for better performance.

2.2 Redis:

Redis (Remote Dictionary Service) is an open source key-value pair database server which is licensed under BSD-license. Accessing the main memory is slow. To reduce this overhead and ultimately completing operations in very small amount of time, Redis works with an in-memory data set. When terminated data will be purged from memory. But depending on the use case, data can be persisted either by taking a snapshot of the data and dumping it on disk periodically or by maintaining an append-only log of all operations. Redis is implemented using ANSI C. Redis architecture is based on BASE approach (Basically Available, Soft-state and Eventually consistent). Redis is based on the client/server architecture and consists following 3 components: 1. Redis server 2. Redis Replica servers (optional) 3. Redis Client. has in-memory, and single threaded architecture. In-memory concept is implemented using the concept of Virtual memory which is a very important feature of most modern operating systems. But for efficiency reasons, Redis does not use the Operating System-supplied Virtual Memory facilities and instead implements its own system called Redis Virtual Memory. The goal of Redis Virtual Memory (VM) is to swap infrequently accessed data from RAM to disk, without

drastically changing the performance characteristics of the database while supporting data sets that are larger than main memory.

2.3 MongoDB:

MongoDB is a document-oriented NoSQL database. The structure used to retrieve and update the database in MongoDb is called BSON, which very similar to JSON. MongoDB compromises availability for data consistency and partition tolerance as defined in the CAP theorem. MongoDB is comprised of three main components. 1. MongoDB server commonly known as mongod daemon 2. MongoDB mongos which is the routing service 3. MongoDB shell. Mongod daemon is the primary process which handles data requests, manages data format, and performs background management operations. There can be many mongod daemons running as primary secondary instances. MongoDB mongos routes information and data in the cluster. MongoDB shell provides an interactive interface. To examine the results of the queries and to check test cases, the developer can use JavaScript commands. In MongoDB, all documents are inserted with a unique hash. This is used when retrieving records while querying. There are several types of other indexes namely, 1. Unique Indexes - applied to a field of a document to reject other duplicate documents 2. Compound Indexes - applied to several fields of a document to reject other duplicate documents 3. Array Indexes - all arrays are indexed by this in a document for performance enhancement 4. TTL Indexes - where it is necessary to remove documents after a certain time to leave 5. Other Indexes - Geospatial Indexes, Sparse Indexes and Text Search Indexes For query optimization MongoDB caches previously run query plans as wells as running several other alternative plans and selects the plan on the best response time. Results of alternative plans are also cached in memory for future reference. MongoDB does operations on data in place and stores its data on contiguous blocks. By updating data in place, it can reduce I/O operations by not requiring movement of data to other locations. It is a known fact that reading from memory is several thousand times faster than reading from Disk, MongoDB uses this to its advantage by keeping frequently accessed indexes in memory.

2.4 ZHT:

ZHT is a zero-hop distributed hash table, adjusted for the specific requirements of high-end computing(for example, trusted/reliable hardware, fast networks, non-existent "churning", low latency, and scientific computing) data access modes) , Designed to be the building block of future distributed systems (high fault tolerance, high performance and scalable storage systems). It uses mature technologies such as TCP, UDP and epoll-based event-driven models, which makes deployment easier. It provides the persistence of the persistent hash table NoVoHT. ZHT can withstand various failures while minimizing overhead. It is also flexible and supports the joining and leaving of dynamic nodes. We have shown that the performance and scalability of ZHT is outstanding with up to 8K nodes and 32K instances. On all 8k and 32k platforms, ZHT

has great potential to be an excellent distributed key-value store and a key building block for large distributed systems.

2.5 Cassandra vs MongoDB vs Redis vs ZHT:

Storage System	Cassandra	MongoDB	Redis	ZHT
Primary database model	Wide column store	Document store	Key-value store	Event-driven model
License	Open Source	Open Source	Open Source	Open Source
Implementation Language	Java	C++	C	C++
Replication Methods	Selectable replication factor	Master-slave replication	Master-slave replication , Multi-master replication	Asynchronous replication
Transaction concepts	No	Multi-document ACID Transactions with snapshot isolation	Optimistic locking, atomic execution of commands blocks and scripts	No
In-memory capabilities	No	Yes	Yes	Yes

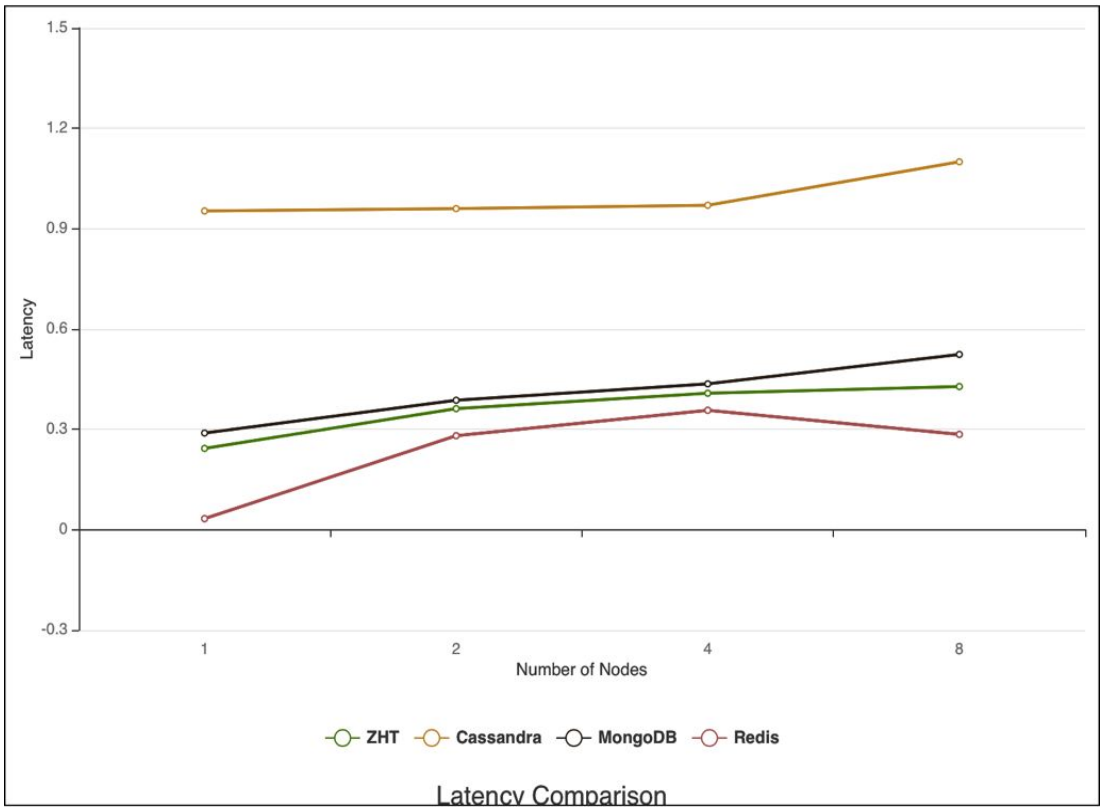
3. Experiments

On each instance/node, a client-server pair is deployed. Test workload is a set of key-value pairs where the key is 10 bytes and value is 90 bytes. Clients sequentially send all of the key-value pairs through a client API for insert. For searching and deleting the key-value pairs the lookup, and remove APIs are used respectively. For one run of the experiments, workload assigned is 300,000 operations i.e 100,000 insert calls, 100,000 lookup calls and 100,000 remove calls. We average across this all 3 operations. The keys are randomly generated, which will produce an All-to-All communication pattern, with the same number of servers and clients. For latency calculations, time taken for every operation as calculated and the average of three operations was considered. On the other hand for throughput calculations, time taken for the

experiments was calculated by taking the difference between time before first insert operation and time after last remove operation.

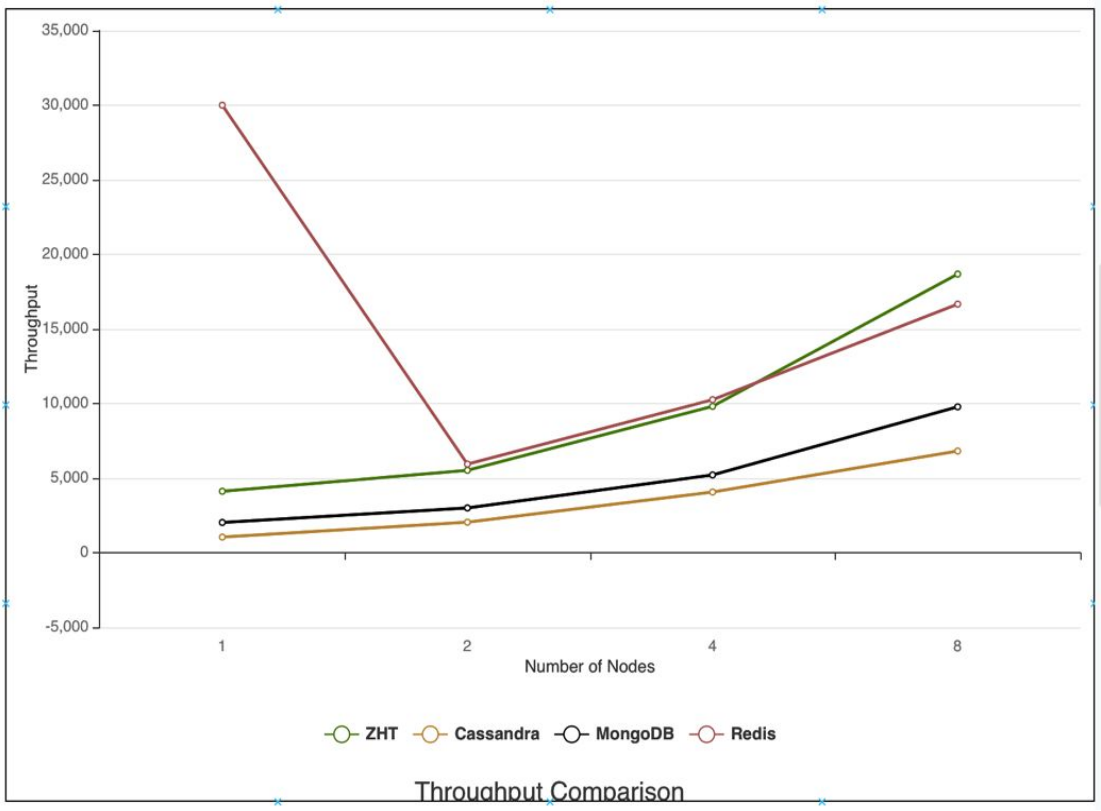
3.1 Latency:

System/Scale	1	2	4	8
ZHT	0.243	0.362	0.408	0.428
Cassandra	0.953	0.960	0.970	1.100
MongoDB	0.289	0.387	0.436	0.524
Redis	0.033	0.281	0.357	0.285



3.2 Throughput:

System/Scale	1	2	4	8
ZHT	4117	5524	9813	18680
Cassandra	1049	2047	4067	6818
MongoDB	2027	3002	5212	9782
Redis	30000	5940	10256	16666



The reason for incredible numbers produced by Redis with one node is that particular experiment was carried out on localhost.

4. Conclusion and Future work

For this weak scaling experiments study we chose Cassandra, MongoDB and Redis. For Cassandra, remove takes the least amount of time and insert operation takes more time

than other operations. On the other hand, MongoDB and Redis take more time to remove than other operations. Also, for MongoDB and Redis, lookup is the fastest operation. After successfully performing the experiments we can conclude that Redis has the best performance as compared to Cassandra and MongoDB in terms of latency and throughput. Results also suggested Cassandra performed the worst. But we still believe, every storage system has their own use-cases. For example, despite being performing worse than Cassandra and MongoDB, Cassandra is the best one suited for applications where high availability is the most important requirement. MongoDB follows document store database model and is the most flexible among these three.

References

[1].Performance Comparison of NoSQL Databases in Pseudo Distributed Mode: Cassandra, MongoDB & Redis Kumarasinghe C.U., Liyanage K.L.D.U. , Madushanka W.A.T. and Mendis R.A.C.L Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka

[2].SQL vs NoSQL: A Performance Comparison Ruihan Wang University of Rochester Zongyan Yang University of Rochester

[3].SQL vs. NoSQL vs. NewSQL- A Comparative Study Sneha Binani Information Technology VESIT Mumbai, India Ajinkya Gutti Computer Engineering VESIT Mumbai, India Shivam Upadhyay Information Technology VESIT Mumbai, India

[4].Storage Performance Virtualization via Throughput and Latency Control JIANYONG ZHANG, ANAND SIVASUBRAMANIAM, and QIAN WANG The Penn State University and ALMA RISKA and ERIK RIEDEL Seagate Research Center

[5]."ZHT: A Light-Weight Reliable Persistent Dynamic Scalable Zero-Hop Distributed Hash Table," 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, Boston, MA, 2013, pp. 775-787