CS 553 PA5

Ali Guzelyel
A20454373
Dapo Ogunnaike
A20435197

Benchmarking Distributed Storage Systems

ZHT, Cassandra, and MongoDB

Technology Review

Cassandra

Apache Cassandra is an open-source distributed storage system designed to manage extensive volumes of structured data globally. It offers high availability services with no single point of failure and excels in scalability, fault tolerance, and consistency. Drawing inspiration from Amazon's Dynamo and Google's Bigtable, Cassandra utilizes a column-oriented database with a distribution design based on Dynamo and a data model influenced by Bigtable.

Cassandra employs the Dynamo-style replication model without a single point of failure, utilizing a robust "column family" data model stored in a single disk file. Organizing columns within the same Column Family is crucial for optimizing performance, and supercolumns can enhance this further. A supercolumn is a specialized column that functions as a key-value pair, storing a mapping of subcolumns. The fundamental data structure in Cassandra is the column, consisting of the key or column name, value, and timestamp.

Designed to handle large data workloads across multiple nodes without a single point of failure, Cassandra is linearly scalable, enhancing throughput with an increasing number of nodes. The peer-to-peer distributed system ensures data distribution across the entire cluster, with each node playing an identical role. In case of node failure, other nodes in the network can handle read/write requests. [ref. Cassandra]

MongoDB

MongoDB, on the other hand, is a distributed system with a document database designed for ease of application development and scaling. MongoDB documents are similar to JSON objects, providing high performance data persistence. The use of embedded documents and arrays reduces the need for expensive joins, and MongoDB's replication facility, called replica set, ensures automatic failover and data redundancy. MongoDB supports horizontal scalability through sharding, has multiple storage engines, and a pluggable storage engine API for third-party development. MongoDB is schema-less, which means that each document in a collection can have different structures. This adds the flexibility to change the data models as the application requirements change over time. MongoDB supports query language along with regular expressions and complex queries. In general, MongoDB is commonly used in applications where flexibility, scalability and development costs are crucial. [ref. MongoDB]

Zero-hop distributed hash table (ZHT) is a tuned hash table implementation tailored for high-end computing requirements. ZHT aims to be a fundamental block for future distributed systems, delivering excellent availability, fault tolerance, high throughput, scalability, persistence, and low latencies. Key features of ZHT include being lightweight, supporting dynamic node joins and leaves, fault tolerance through replication, customizable consistent hashing function, persistence for recoverability, scalability, and unconventional operations alongside traditional hash table operations. [ref. ZHT]

Comparisons

ZHT has not been put into industry yet, so it is exempt from the comparisons below. However, we will compare all of them with respect to Distributed Systems criterias such as Performance(Throughput-Latency), Scalability, and Accessibility/Flexibility.

MongoDB	Cassandra
● Document-oriented database. ● Stores data in BSON (Binary JSON) documents. ● Supports flexible schemas.	● Wide-column store database. ● Organizes data in tables with rows and columns. ● Schemas are defined per query, providing flexibility
● Horizontal scaling through sharding. ● Provides automatic sharding to distribute data across clusters.	● Designed for distributed architecture. ● Linearly scalable across multiple nodes and clusters.
● Provides tunable consistency, allowing users to balance between consistency and availability. ● Configurable write concerns.	● Provides tunable consistency levels (QUORUM, ONE, ALL, etc.). ● Designed to ensure high availability and fault tolerance.
● Well-suited for applications requiring flexibility in data models, such as content management systems, real-time analytics, and catalogs.	● Ideal for time-series data, IoT applications, and scenarios with high write throughput and scalability requirements.
● Can be deployed on a single server or as a distributed system. ● Uses a master-slave architecture by default, with automatic failover.	● Distributed by design, with no single point of failure. ● Employs a peer-to-peer architecture, providing fault tolerance and scalability

Table 1: Comparing Cassandra and MongoDB based on industry usage criterias

Technology Name	Performance	Scalability	Accessibility/Flexibility
ZHT	1	1	2
Cassandra	2	3	3
MongoDB	3	2	1

Table 2: Ranking ZHT, Cassandra, and MongoDB on performance, scalability, accessibility

Technolog y Name	Implementatio n	Routing Time	Persistence	Dynamic Membership	Append
ZHT	C++	0 to 2	Yes	Yes	Yes
Cassandra	Java	log(N)	Yes	Yes	No
MongoDB	C++	log(N)	Yes	Yes	No

Table 3: Comparison between ZHT, Cassandra, and MongoDB [ZHT-2]

The technologies mentioned above are all NoSQL databases. Cassandra is based on a wide-column store based on key-value pairs, whereas MongoDB is based on document type, and ZHT uses FusionFS file system. ZHT paper shows that it scales almost perfectly compared to other systems, with MongoDB supporting a flexible scaling option which puts it at a better rank than Cassandra. ZHT has a brilliant routing time of constant 0-2 ms, whereas Cassandra and MongoDB has linear routing times. All systems are persistent and supports dynamic membership. ZHT also supports lock-free concurrent key-pair modifications, as shown in the table under *append*.

Benchmarking Procedure

(Another description of the procedure and installations are given in the README.md file.)

Test Procedure

We are testing for insert (write), lookup (read), remove (delete) operations. We will take into account the durations for all operations and the throughput calculated from the tests with single and multiple cluster operations.

Test Dataset

We generated a dataset of 1000 rows that has a key of 10 bytes and value of 90 bytes. The keys and the values are random. To generate the keys and values for the dataset, run: generate-all-data.sh, which will run automatically generate-keys.sh, generate-values.sh, and generate-random-pairs.py.

Chameleon

Lease an instance from https://chameleoncloud.org/ and create a compute instance with compute_skylake. Attach a Floating IP address. SSH to the instance, and create 8 Virtual Machines (3-core, 12gb memory, 24gb storage) with the generate-small-instances.sh. Add configurations to UFW to allow IPV4 connections (included in generate-small-instances.sh). Connect all instances with the code in connect-instances.sh (some manual configurations may be required). In addition, small-instance-1 needs to have ip addresses of all other clusters by adding them to the /etc/hosts file.

Cassandra

Setup Cassandra on all small-instances by running setup-cassandra.sh along with run-script-on-all-VMs.sh. This will first install JAVA JDK on each cluster, and then set up

Cassandra 4.0.11. After the setup, we configure the cassandra.yaml file to include the IP addresses on rpc_address, listen_address, and seed_provider. After all configurations are done, we can start cassandra by running: bin/cassandra. To check status, run: bin/nodetool status. Install python for dataset generation and cassandra benchmark tests. To stop the cassandra servers on all clusters, run: sh run-script-on-all-VMs.sh stop-cassandra.sh

MongoDB
Setup MongoDB on all small-instances by running: sh run-script-on-all-VMs.sh setup-mongodb.sh. This will install mongodb on all clusters, and configure for default configurations. It will also configure cluster limitations according to mongodb suggestions. We manually configured the mongod.conf file to add the IP addresses and replication set name. Allow the 27017/default TCP port, and enable mongod on systemctl.
After the configurations are done, we can start mongod on all clusters with systemctl restart mongod.
We also need to install mongocxx to run C++ code on the MongoDB server.

Benchmarking Results

Insert Latency (ms)

Scale	**1**	**2**	**4**	**8**
Cassandra	0.886	0.516	0.351	0.014
MongoDB	0.191	0.539	0.670	0.679

Lookup Latency (ms)

Scale	**1**	**2**	**4**	**8**
Cassandra	0.988	0.326	0.551	0.472
MongoDB	0.128	0.501	0.610	0.639

Remove Latency (ms)

Scale	**1**	**2**	**4**	**8**
Cassandra	0.423	0.432	0.739	0.787
MongoDB	0.123	0.298	0.339	0.397

Average Latency (ms)

Scale	**1**	**2**	**4**	**8**
Cassandra	0.766	0.425	0.547	0.424

MongoDB	0.147	0.446	0.539	0.571

Insert Throughput

Scale	**1**	**2**	**4**	**8**
Cassandra	1128	3875	11396	571428
MongoDB	5235	3710	5970	11782

Lookup Throughput

Scale	**1**	**2**	**4**	**8**
Cassandra	1012	6135	7259	16949
MongoDB	7812	3992	6557	12519

Remove Throughput

Scale	**1**	**2**	**4**	**8**
Cassandra	2364	4629	5412	10165
MongoDB	8130	6711	11799	21622

Average Throughput

Scale	**1**	**2**	**4**	**8**
Cassandra	1305	4705	7312	18868
MongoDB	6802	4484	7421	14011

Benchmarking Analysis

In this experiment, we benchmarked Cassandra and MongoDB distributed systems on key-value type insert, read, and remove operations.
Comparisons: Overall MongoDB is faster than Cassandra in operations with one cluster. MongoDB experiences a drop in performance as soon as we start using two clusters, but reaches a well performance threshold quickly with four and eight clusters. Comparing latencies, we can see that insert operations have less latency for Cassandra than MongoDB, which could be because of MongoDB’s lazy execution design. Similar results are seen in read operations. However, remove operations are highly faster for MongoDB. After four clusters, both systems reach a linear throughput performance. As we increase the number of clusters, the throughput increases as much as the increase in clusters. This could be because the system reaches a good latency performance by four cluster systems.

In general, MongoDB and Cassandra are both good at respective aspects, and they are both good at optimization with large number of clusters, which is important for scaling as distributed systems need large number of clusters to store and process Big Data.

References

[Cassandra] https://cassandra.apache.org/_/index.html
[MongoDB] https://www.mongodb.com/
[ZHT] http://datasys.cs.iit.edu/reports/2012_Qual-IIT_ZHT.pdf
[ZHT-2] Tonglin Li, Xiaobing Zhou, Kevin Brandstatter, Dongfang Zhao, Ke Wang, Anupam Rajendran, Zhao Zhang, Ioan Raicu. “ZHT: A Light-weight Reliable Persistent Dynamic Scalable Zero-hop Distributed Hash Table”, IEEE International Parallel & Distributed Processing Symposium (IPDPS) 2013;
http://datasys.cs.iit.edu/publications/2013_IPDPS13_ZHT.pdf