CS553 – Cloud Computing. Homework 7

Authors:

Luis Cavanillas (A20474430)
Jorge Cervera (A20474346)
Jorge Gonzalez (A20474413)

Semester: Spring 2021

Problem Statement

Various distributed key/value storage systems have been evaluated and compared with the existing literature. In specific, Cassandra and MongoDB have been the ones evaluated and compared with the results found in [1], of which the systems ZHT, Cassandra and Memcached have been subtracted.

Proposed Solutions

The systems chosen for this assignment are Cassandra and MongoDB and their key features are going to be analyzed and compared with ZHT.

1. Cassandra: It is a distributed NoSQL database management system that is built to handle large amounts of data across multiple data centers and the cloud. Their main key features include that it is highly scalable (its structure allows users to meet sudden increases in demand, as it allows users to simply add more hardware to accommodate additional customers and data), offers high availability and has not a single point of failure (it offers truly consistent access and availability). It is one of the most efficient and widely used NoSQL databases. At the same time, it allows a flexible data storage (it can handle structured, semi-structured, and unstructured data), flexible data distribution (it uses multiple data centers, which allows for easy data distribution) and supports ACID (atomicity, consistency, isolation, and durability).
   At the same time, Cassandra offers flexibility to create columns within rows (schema-optional data model) and since each row may not have the same set of columns, there is no need to show all the columns needed by the application at the surface. With Cassandra each node in the cluster has the same role and the data set is distributed across the whole cluster and it supports Hadoop integration with MapReduce support. Also, for accessing Cassandra, the language CQL (Cassandra Query Language) has been introduced.

2. MongoDB: It is also a scalable and flexible NoSQL document database platform designed to overcome the relational databases approach and the limitations of other NoSQL solutions. It also offers horizontal scaling, flexibility and scalability with a load balancing capability.
   Some of its most important features are the following: it supports ad hoc queries (data can be searched by field, range query and regular expression searches), any field in a

document can be indexed, it supports Master Slave replication (a master can perform reads and writes and a slave copies data from the master and can only be used for reads or back up), it can run over multiple servers and the data is duplicated to keep the system up and also keep its running condition in case of hardware failure.
At the same time, it has an automatic load balancing configuration, based on data distribution in shards. This operation is handled through a lightweight process called mongos. Mongos can direct queries to the correct shard based on the shard key.

3. ZHT: Zero-hop distributed hash table (ZHT),is another distributed system that aims to deliver excellent availability, fault tolerance, high throughput, scalability, persistence, and low latencies. ZHT has several important features making it a better candidate than other distributed hash tables and key-value stores, such as being light-weight, dynamically allowing nodes join and leave, fault tolerant through replication and by handling failures gracefully and efficiently propagating events throughout the system, a customizable consistent hashing function, supporting persistence for better recoverability in case of faults, scalable, and supporting unconventional operations such as append (providing lock-free concurrent key/value modifications) in addition to insert/lookup/remove.

All differences between features are summarized in Table 1.

Feature	**Cassandra**	**MongoDB**	**ZHT**
Areas of use	Banking, finance, logging	CMS system, comment storage	High-end computing
Structure	Combines tabular and key/value properties.	Documents	Key-Value
Schema	Data structure is static. Column type and categories must be defined from the start	Does not require schemas	Hash table (NoVoHT)
Replication	Multi-Master	Master-Slave	Multi-Master
Syntax	Cassandra Query Language (CQL)	MongoDB Query Language (MQL)	C++
CAP theorem	Partition tolerance, High Availability	Consistency, Partition tolerance	Consistency, High availability
Concurrency	MVCC	Instant update	Metadata lookups
Clustering	No master-slave architecture.	Mongo Master, Mongo Shard, and Mongo Config.	-
Sharding	Partition Key, Composite Key, Clustering Key (columns)	Use of indexes.	-

Table 1. Summarized comparison between Cassandra, MongoDB and ZHT technologies.

Cassandra deployment.

Cassandra has been installed and configured (by editing ***/etc/cassandra/cassandra.yaml*** file) in the 8 nodes (virtual machines) separately. Cassandra cluster has been configured with every node being a seed of the rest of the nodes. This configuration allows one client/server pair per node.

The execution of the program will be run in each of the nodes (virtual machines), after a *parallel ssh* command executed in the Chameleon instance. This command will order all the hosts listed in *pssh_hosts_files* list (previously configured with the 8 nodes' addresses) to execute cassandraDB.py.

parallel-ssh -i -h ~/.pssh_hosts_files python3 cassandraDB.py

When running the program, each node will perform the required operations (insert/look up/remove) its corresponding number of records (10,000 records/number of nodes) over the distributed database. Note that this configuration has been edited for every case, in 1, 2, 4 and 8 nodes experiments.

MongoDB deployment.

As Cassandra, MongoDB has been installed and configured in all the nodes. The configuration has been made by editing **/etc/mongod.conf** file.

Each node will be configured independently. This is required as with the topology of MongoDB only the master has the permissions to perform write operations and the slaves can only do read ones. That is also the reason why the lookup operations are so fast in MongoDB.

We will deploy a cluster with this topology in which all the nodes will perform the required actions over the database. This configuration is not the pure Sharded Cluster configuration but has a similar performance in terms of throughput and latency over these experiments and assures that every node can write and read the database records.

Similar to Cassandra's deployment, our program will be run with a parallel-ssh command which will execute the operations in all the nodes simultaneously:

parallel-ssh -i -h ~/.pssh_hosts_files python3 mongoDB.py

The nodes configuration has been edited for every case, in 1, 2, 4 and 8 nodes experiments.

Results

The scripts made for the Cassandra and MongoDB performance benchmark have been both developed using Python3.7 for the sake of simplicity and lack of time. The imports of both drivers in these programming languages simplifies a lot the coding process compared to coding in other languages such as Java (external dependencies, jar files, etc.) or C/C++ (external

libraries, third-party codes, etc.). Moreover, the complexity of both codes is negligent, thus no performance bottlenecks are present by using Python.

Type Op. /Scale	1	2	4	8
Insert	0.512711	0.5992786	0.824123	0.982732
Lookup	0.61407	0.725903	0.974322	1.175676
Delete	0.420654	0.4893	0.682731	0.893822

Type Op./Scale	1	2	4	8
Insert	0.201568	0.214875	0.231039	0.312852
Lookup	0.003651	0.005684	0.007618	0.010328
Delete	0.46676	0.54608	0.557365	0.560522

Table 2. Performance evaluation of Cassandra ((A) upper table) and MongoDB ((B) lower table) plotting type of operation vs. scale (1 to 8 nodes); unit of measurement is milliseconds per operation.

System/Scale	1	2	4	8
Cassandra	0.51581	0.60483	0.82706	1.01741
MongoDB	0.22399	0.25555	0.26534	0.29457

System/Scale	1	2	4	8
ZHT	0.24300	0.36200	0.40800	0.42800
Cassandra	1.19900	1.87000	1.87500	1.99400
Memcached	0.12200	0.32400	0.27200	0.27800

Table 3. Performance evaluation plotting system vs. scale (1 to 8 nodes); unit of measurement is milliseconds per operation. (A) Upper table - own benchmark (B) Lower table - benchmark in [1].

System/Scale	1	2	4	8
Cassandra	1939	3307	4836	7863
MongoDB	4464	7826	15075	27158

System/Scale	1	2	4	8
ZHT	4117	5524	9813	18680
Cassandra	926	1131	2189	4322
Memcached	7385	7961	14480	40995

Table 4. Throughput evaluation plotting system vs. scale(1 to 8 nodes); unit of measurement is operations per second. (A) Upper table - own benchmark (B) Lower table - benchmark in [1].

As shown in Table 2 and Table 3, MongoDB performs better than Cassandra in terms of speed. Theoretically, Cassandra performs better in applications that require heavy data load since it can support multiple master nodes in a cluster. MongoDB would not be ideal for applications with heavy data load as it can't scale with the performance. Therefore, as this exercise involves querying 10.000 rows (considered a small amount of data), MongoDB still performs better. If millions of rows of data, each one of thousands of bytes, were used, probably Cassandra would have performed better.

All in all, MongoDB provides 2x or more greater throughput in mixed workloads compared to Cassandra.

Comparing Table 4.A with Table 4.B, MongoDB behaves similarly to ZHT, or even a little bit better when more nodes are used. Memchached still shows the best performance results after this exercise has been completed.

References:

[1] 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.