SEIS630: Final Project Report
Benchmarking Cassandra Using Yahoo's Cloud Serving Benchmark (YCSB)

I. Abstract

Benchmarking databases is a key step in the software industry as it helps to select the most suitable long-term database technology for the task at hand. In a similar spirit, this work studies a popular NoSQL database Cassandra in a distributed environment through a workload based performance analysis. The experiments are performed for different data replication and data partitioning strategies using Yahoo's Cloud Serving Benchmark (YCSB). The benchmark provides functionality to generate synthetic datasets which are queried through custom workloads. Each workload varies according to the type of executed operation (i.e., read, scan, update, write, and insert). The performance of each workload is measured using throughput, latency, and runtime. The results provide key insights into the effect of different replication/partitioning strategies used during the database configuration.

II. Introduction to NoSQL databases

Traditional database management systems (DBMS) that are based on the relational model (SQL databases), have proven to be reliable for a wide range of transaction-based applications on structured data. Most of this data is in unstructured formats such as videos, graphs, speech, websites and blogs, sensors in the world of Internet of Things (IoT), social media posts, etc. Most of it is used for analytical and non-transactional operations. Here, the NoSQL data stores are more appropriate with their capabilities to deal with native forms such as documents, graphs, time-series, and multimedia files.

NoSQL databases ("not only SQL") are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide flexible schemas which scales relatively easily with large amounts of data and higher user loads. This shift to NoSQL database attributes to two functional requirements: first, the growing needs for the data flexibility and availability over data consistency and second, the emphasis on scaling-out over scaling-up of the hardware resources. In a scale-out system, new hardware can be added and configured on demand whereas, in a scale-up system, the existing hardware is replaced with better configurations.

Fundamentally, a centralized database system is required to provide consistency. When it comes to a distributed database system, the CAP, (Consistency, Availability, and Partition tolerance), theorem defined by Fox and Brewer [5, 6], states that a distributed system can only meet two out of the three needs simultaneously.

This work evaluates the performance of a popular NoSQL database Cassandra.

III. Cassandra:

Cassandra was developed at Facebook as a combination of Amazon's DynamoDB and Google's Bigtable. Cassandra is a distributed system where each node plays two roles, master and a slave. The system performs all the critical functions in a decentralized manner. It offers elastic scalability, high availability, tunable consistency along with Cassandra query language (CQL). The nodes communicate with each other using a peer-to-peer communication protocol in which nodes periodically exchange state information about themselves and about other nodes they know about. This protocol is known as Gossip protocol.

1. Key structures: Basic structures used in Cassandra are Node, Datacenter, Cluster, Commit log, SSTable, CQL table, Rack [3, 4].

- Node: The basic infrastructure component where data is stored.
- Datacenter: A collection of related nodes. It can be either virtual or physical.
- Cluster: A cluster contains one or more data centers. It can span multiple physical locations.

- Commit log: All data is written first to the commit log for durability.
- SSTable: A sorted string table which is an immutable data file to which Cassandra writes memtables periodically.
- CQL Table: A collection of ordered columns fetched by the table row. A table consists of columns and has a primary key.

2. Gossip protocol: A peer-to-peer communication protocol to discover and share location and state information about the other nodes in a Cassandra cluster. Gossip information is persisted locally by each node to use immediately when a node restarts. It can be used with multi-node clusters spanned across multiple data centres. From a higher level, Cassandra's single and multi data center clusters are described in the figure below (Figure 1).

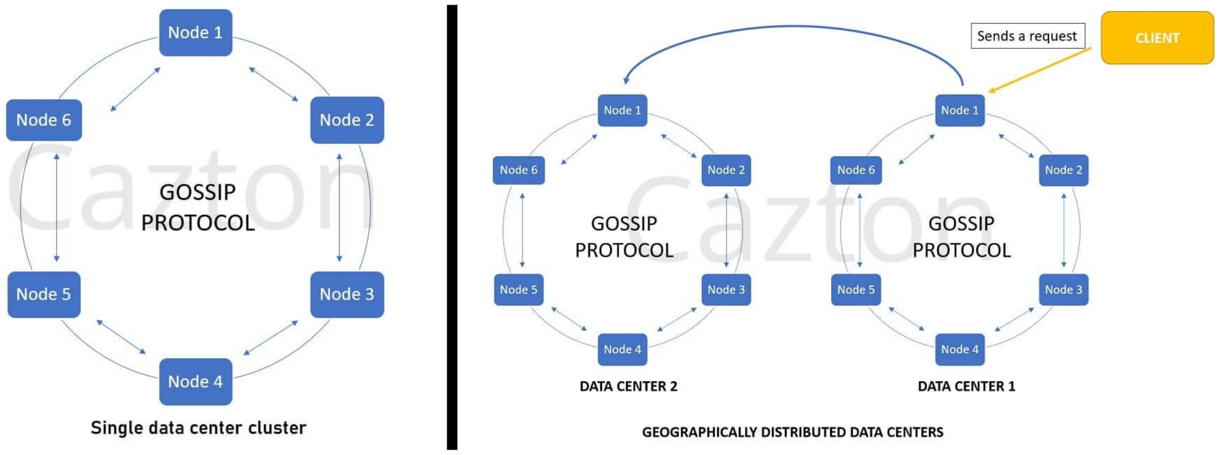

Figure 1. Gossip protocol in single and multiple Cassandra data centres [2].

2. Data distribution and replication: In cassandra, data distribution and replication is required simultaneously. A partitioner determines how the data is distributed across the nodes in the cluster (including replicas). It stores replicas of each row on multiple nodes to ensure reliability and fault tolerance.

Partitioning Strategies: A partitioner is a hashing function which derives a token hash value for each row using its partition key. Each data row is then distributed across the cluster by the value of the token. Three types of partitioners are defined in Cassandra.

- Murmur3Partitioner (default) : This type of partitoning uniformly distributes data across the cluster based on MurmurHash hash values. A MurmurHash function creates a 64-bit hash value of the partition key. The possible range of hash values is from -2^{63} to $+2^{63}-1$.
- RandomPartitioner: This type of partitioner uniformly distributes the data across the cluster using MD5 hash values. The possible range of hash values is from 0 to 2^{127} -1.
- ByteOrderedPartitioner: This keeps an ordered distribution of the data lexically by its key bytes.

The main difference across different partitioning strategies is how each partitioner generates the token hash values. The RandomPartitioner uses a cryptographic hash that takes longer to generate than the Murmur3Partitioner. Cassandra doesn't need a cryptographic hash, so using the Murmur3Partitioner results in a three to five times improvement in the query performance.

Data Replication: A replication strategy determines the nodes where the replicas are stored. The total number of replicas across the cluster is referred to as the replication factor. A replication factor of 1 means that there is only one copy of each row in the cluster (i.e., no duplication). If the node containing the row goes down, the row cannot be retrieved. Two replication strategies are available:

- SimpleStrategy (default): This strategy is used only for a single datacenter and one rack. It places the first replica on a node determined by the partitioner. Additional replicas are placed on the next nodes clockwise in the ring.
- NetworkTopologyStrategy: This strategy is used when the cluster is deployed across multiple datacenters. It specifies how many replicas should be made in each datacenter. This strategy attempts to place replicas on distinct racks because nodes in the same rack often fail at the same time due to power, cooling, or network issues.

3. Snitch: A snitch defines a group of machines into data centers and racks that the replication strategy uses to place replicas. This property should be configured when a cluster is created. Default is SimpleSnitch which is used for a single datacenter.
4. Cassandra.yaml configuration file: The main configuration file for setting the initializing properties for a cluster, parameters for tables, properties for tuning and resource utilization, timeout settings, client connections, backups, and security.
5. System keyspace table properties: The storage configuration attributes are set on a per-keyspace or per-table basis programmatically or using client applications, such as Cassandra Query Language (CQL).

IV. Case Study

This work studies the Cassandra database in a distributive environment through a systematic performance analysis. This analysis is conducted using *SimpleStrategy* as data replication strategy and *Murmur3Partitioner* as partitioning strategy and using the datasets generated by YCSB. The overall throughput and latency is measured with respect to the following atomic operations: read, insert, update, read-modify-write, and scan for each of the six workloads.

YCSB Framework: Yahoo under Yahoo's Cloud Serving Benchmark (YCSB) [8] project developed a framework and a common set of workloads to evaluate the performance of different data stores. The project has two key components. First, the YCSB client that is an extensible workload generator. Second, the core workloads which are a set of workload scenarios to be executed by the generator.

The Client is extensible for defining new and different workloads to help examine various system aspects or application scenarios. The core workloads provide a well rounded picture of a given system's performance.

A common use of the tool is to benchmark multiple systems and compare them on the same hardware configuration. However, for this analysis the performance analysis was done for one data store i.e., Cassandra.

Workloads Description: The workload defines the data that will be loaded into the database during the loading phase, and the operations that will be executed against the data set during the run phase. Each workload goes through two phases, the "LOAD" phase in which random data is generated and written in the data-store, and the "RUN" phase in which atomic operations are performed on the written data.

The six workloads provided by YCSB are composed of one or more of the following atomic query operations: read, insert, update, read-modify-write, and scan (read up to 100 consecutive rows in the database from a random start point). The relative frequency of each operation is defined in the parameter file, as well as other properties of the workload. The parameter file can be modified to execute a variety of workloads.

Technically, a workload is a combination of:

- Workload java class (subclass of com.yahoo.ycsb.Workload)

- Parameter file (in the Java Properties format)

The same property file is used in both the phases so that a similar set of records can be constructed and inserted during loading phase, and then subsequently used during the run phase. The workload compositions are described in Table 1.

Table 1. Workloads composition for different types of requests.

Workload	Composition
Workload A	50% Read + 50% Update
Workload B	95% Read + 5% Update
Workload C	100% Read
Workload D	95% Read + 5% Insert
Workload E	95% Scan + 5% Insert
Workload F	50% Read + 50% Read-Modify-Write

Different runtime parameters can be defined during the execution of a workload on the YCSB client command line. These settings are:

- -threads: The number of client threads. By default, the YCSB client uses a single worker thread, but additional threads can be specified using this property.
- -target: The target number of operations per second. To generate a latency versus throughput curve, different target throughputs can be tried and latency for each can be measured.
- -s: The status record of load or run phase for workload. By specifying “-s” on the command line, the Client will report status every second to stderr or a separate log file.

Metrics for Measurement: The evaluation results are mainly based on latency and throughput based metrics.

- Latency: The latency metric represents the time interval between the function call and the response to the call.
- Throughput: This metric represents the rate of successful execution of a functional call.

Two different approaches are followed for “LOAD” and “RUN” phases. In the “LOAD” phase, multiple YCSB instances run in parallel. The throughput is measured as the addition of throughput measures from all the instances and average of latency measures from all the instances. In the “RUN” phase, a single instance is used whose output measures are then reported.

IV. Experimental Setup

The experiments are conducted on single node cluster on Microsoft Windows 10 (Version 10.0.19044) machine with the following processor configuration Intel(R) Core(TM) i7-8550U CPU @1.80GHz, 1992 Mhz, 4 Core(s), 8 Logical Processor(s).

- **Steps for YCSB installation**
    1. Download the latest release of YCSB [9] using the command below:
       ```
       Curl -O --location
       https://github.com/brianfrankcooper/YCSB/releases/download/0.17.0/y
       csb-0.17.0.tar.gz
       tar xfvz ycsb-0.17.0.tar.gz
       cd ycsb-0.17.0
       ```
       The YCSB version used for this analysis is ycsb-0.17.0.
    2. Verify the installation directory and add it to the classpath.

3. Run ‘ycsb’ command to check the installation and check the usage.
4. Set up a Cassandra database to benchmark following the instructions in the ReadMe file under Cassandra database binding. (Additional details below)
5. Run each of the workloads with ‘load’ and ‘run’ commands on ycsb client.

- **Steps for Cassandra installation**
    1. Download the latest Apache Cassandra 2.x CQL binding from cassandra.apache.org [10]. The last stable Cassandra version 3.11.11 was used for this experiment.
    2. Verify the integrity of the downloaded files by using their PGP signature (.asc file) or a hash (.md5 or .sha* file) [11].
    3. Extract the zipped/tar version into the installation directory.
    4. Add CASSANDRA_HOME=\installation directory\ to the environment variables.
    5. From the cassandra \install directory\bin path, open the command line and execute ‘cassandra’ command to start the server.
    6. Verify the ‘startup complete’ message for a successful server startup on localhost:9042.

- **Steps for starting CQL command line in cassandra.**
    1. From the cassandra installation directory\bin open a new command line and execute ‘cqlsh’ command.
    2. Cqlsh.bat file is executed and CLI is connected to Cassandra port.
    3. Create a ycsb keyspace and a table for use with YCSB on cqlsh command line as shown below:
```
cqlsh> create keyspace ycsb
WITH REPLICATION = {'class' : 'SimpleStrategy',
'replication_factor': 3 };
cqlsh> USE ycsb;
cqlsh> create table usertable (
    y_id varchar primary key,
    field0 varchar,
    field1 varchar,
    field2 varchar,
    field3 varchar,
    field4 varchar,
    field5 varchar,
    field6 varchar,
    field7 varchar,
    field8 varchar,
    field9 varchar);
```

- **Steps for running workloads**
    1. Navigate to the ycsb installation directory and start a new command line.
    2. Run the basic workload to verify the installation using the below command.
```
$ ./bin/ycsb load basic -P workloads/workloada
Verify the console output:
Loading workload…
Starting test.
```
    3. Execute load phase for all six workloads with ‘load’ command and record the output in a separate log file.
```
./bin/ycsb load cassandra-cql -p hosts=127.0.0.1 -P
workloads/workloada -s > load_workloada.txt
```

4. Execute run phase for all six workloads with 'run' command and record the output in a separate log file.
```
./bin/ycsb run cassandra-cql -p hosts=127.0.0.1 -P
workloads/workloada -s > run_workloada.txt
```

V. Results

Table 2, Table 3, and Table 4 describe the metrics that were recorded for both load and run phase on a single node cluster.

Table 2. Performance metrics for Load phase of six workloads.

	Metrics	**Workloada**	**Workloadb**	**Workloadc**	**Workloadd**	**Workloade**	**Workloadf**
Overall	Runtime(ms)	6467	6306	6207	6320	6279	6278
	Throughput(ops/sec)	154.63	158.57	161.108	158.22	159.26	159.28
Insert operation	Operations	1000	1000	1000	1000	1000	1000
	Average Latency(us)	3491.139	3426.92	3335.777	3397.795	3357.364	3384.075
	Militancy(us)	1005	895	963	978	992	934
	Militancy(us)	40031	18847	21327	17983	19711	18863
	95thPercentileLatency(us)	4391	4491	4291	4303	4303	4343
	99thPercentileLatency(us)	5299	5759	5215	5219	5215	5179
	Return=OK	1000	1000	1000	1000	1000	1000

Table 3. Performance metrics for the run phase of Workload A, Workload B, Workload C

	Metrics	**Workloada**	**Workloadb**	**Workloadc**
Overall	RunTime(ms)	6335	6194	6230
	Throughput(ops/sec)	157.853197	161.44656	160.513644
Read	Operations	511	947	1000
	AverageLatency(us)	3502.91389	3278.6737	3349.18
	MinLatency(us)	773	891	881
	MaxLatency(us)	24047	19279	19087
	95thPercentileLatency(us)	4523	4283	4743
	99thPercentileLatency(us)	5475	5199	5919
	Return=OK	511	947	1000
Update	Operations	489	53	NA
	AverageLatency(us)	3382.35174	3810	NA
	MinLatency(us)	835	1874	NA
	MaxLatency(us)	12807	7195	NA
	95thPercentileLatency(us)	4287	6127	NA
	99thPercentileLatency(us)	6787	6963	NA
	Return=OK	489	53	NA

Table 4. Performance metrics for the run phase on Workload D, Workload E, and Workload F.

	Metrics	**Workloadd**		**Workloade**		**Workloadf**
Overall	RunTime(ms)	6189	**Overall**	6674	**Overall**	7584
	Throughput(ops/sec)	161.576991		149.83518		131.85654
Read	Operations	955	**Scan**	949	**Read**	1000
	AverageLatency(us)	3285.64817		3771.5068		3161.576
	MinLatency(us)	852		1114		825
	MaxLatency(us)	18559		33599		19247
	95thPercentileLatency(us)	4355		5091		4203
	99thPercentileLatency(us)	5155		6415		4483
	Return=OK	955		949		1000
Insert	Operations	45	**Insert**	51	**Update**	484
	AverageLatency(us)	3549.82222		3304.9216		3124.77066
	MinLatency(us)	3549.82222		1170		888
	MaxLatency(us)	12231		6411		10375
	95thPercentileLatency(us)	5131		4859		4171
	99thPercentileLatency(us)	12231		5347		5815
	Return=OK	45		51		484

Workload A: Update heavy workload
This workload has 50% read and 50% update requests. The total execution time taken is 6.33 seconds. The average throughput is 156.8 operations per second across all threads. In a single node cluster, the average latency for 511 read requests is 3502.91 micro seconds. The average latency for 489 update requests is 3382.35 micro seconds.

Workload B: Read mostly workload
This workload has 95% read operations and 5% update operations. The total execution time taken is 6.194 seconds. The average throughput is 161.44 operations per second across all threads. In a single node cluster the average latency for 947 read requests is 3298.67 microseconds and average latency for 53 update requests is 3810 micro seconds.

Workload C: Read only
This workload has 100% read requests. The total execution time taken is 6.23 seconds. The average throughput is 160.51 operations per second across all threads. In a single node cluster, the average latency is 3349.18 micro seconds for 100 read requests.

Workload D: Read latest record
In this workload, new records are first inserted and then the latest ones are read. It has 95% read requests and 5% insert requests. The total execution time taken is 6.189 seconds. The average throughput is 161.57 operations per second across all threads. In a single node cluster, the average latency is 3285.64 microseconds for 955 read requests and 3549.82 microseconds for 45 insert requests.

Workload E: Short ranges
This workload has 95% scan requests and 5% insert requests. In this workload, short ranges of records are queries instead of individual records. The total execution time taken is 6.67 seconds. The average throughput is 149.83 operations per second across all threads. The average latency for 949 scan requests is 3771.5 microseconds and 3304.92 microseconds for 51 insert requests.

Workload F: Read-modify-write
In this workload, the record is read first, then modified and finally changes are updated. This enforces the read operation to be executed first before any write operation on the same record. This workload has 50%

read and 50% read-modify-write requests. The total execution time taken is 7.584 seconds. The average throughput is 131.85 operations per second. The average latency for 1000 read requests is 3161.57 microseconds and 3124.77 microseconds for 484 update requests.

VI. Conclusion

The overall throughput of approxmiately 161 operations per second is the highest for read heavy workloads (Workload B, Workload C, Workload D). This could be due to negligible network delay as these workloads are tested on a single node cluster. Thus, Cassandra seems to be more suitable for read intensive workloads on a single node.

Cassandra performs good for update heavy workload (Workload A) in terms of latency as the average latency for 489 update requests is 3382.35 microseconds which is close to average latency for just 53 update requests in a read mostly workload (Workload B) because it optimizes its write operations by making the data immediately available on a single node and progressively updates the other nodes. Although, this small margin could be attributed to the high number of read requests in Workload B.

Finally, Cassandra's performance is highly dependent on the way the data model is designed. While designing a Cassandra data model the most important rules to follow are, first is to spread the data evenly in the cluster, which means having a good primary key. Second, reduce the number of partition reads. Cassandra is recommended for read heavy operations as it provides steady data availability and doesn't have a single point of failure because it has replicas stored on numerous nodes. Still, cassandra is also a good fit for write operations as it avoids random data input and has a mechanism to restore all the lost in-cache writes.

VII.References:

[1] https://www.scnsoft.com/blog/cassandra-performance
[2] https://cazton.com/consulting/database-development/cassandra
[3] https://docs.datastax.com/en/landing_page/doc/landing_page/cassandra.html
[4]https://cassandra.apache.org/doc/latest/
[5] E. Brewer. A certain freedom: thoughts on CAP theorem.In proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing, pages 335-335. ACM, 1984.
[6] A. Fox and E. A. Brewer. Harvest, yield, and scalable tolerant systems. In Hot topics in Operating Systems, 1999. In proceedings of the Seventh workshop on, pages 174-178 IEEE, 1999.
[7] Abdeltawab Hendawi, Jayant Gupta, Jiayi Liu. Distributed NoSQL Data Stores: Performance Analysis and a Case Study, In 2018 IEEE International Conference on Big Data (Big Data), pp. 1937-1944. IEEE, 2018. (Industry and Government Track). [Link]
[8] 2008. [12] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In Proceedings of the 1st ACM symposium on Cloud computing, pages 143–154
[9] YCSB releases.https://github.com/brianfrankcooper/YCSB/releases/tag/0.17.0
[10] cassandra.apache.org.https://cassandra.apache.org/_/download.html
[11] apache.org.https://www.apache.org/dyn/closer.lua/cassandra/3.11.11/apache-cassandra-3.11.11-bin.tar.gz.