Scaling and Benchmarking

MongoDB vs Cassandra using YCSB

Bruno Madureira, Nuno Miranda, Vasco Patrício
Department of Informatics Engineering, FCTUC
Coimbra, Portugal.

***Abstract*—Performance and scalability are key properties of database systems. We need to study these properties in order to choose the DBMS that will allow us to overcome the engineering challenges we might run into. This paper aims to benchmark the databases MongoDB and Cassandra in similar working and scaling scenarios. The goal is to measure in which segment a system is better than the other, if it is in writing, reading, etc. We will try to justify the performance differences and try to see in which scenarios the use of one database is more adequate than the other. In order to do this we'll formulate hypothesis and test them in a formal way, using YCBS as a benchmarking tool. We concluded that our system does not represents a real system, so we can't conclude more than which system works best in this scenario, and the winner was MongoDB in a single node scenario.**

I. INTRODUCTION

Choosing the right DBMS for you it's not an easy task. Nowadays there are a lot of options and no absolute answer. In this paper we will look into two very popular NO-SQL DMBS, Apache Cassandra and MongoDB and try to give an overview of each one's weaknesses and strengths. Our analysis will find out between this two DBMS, who behaves better under different workload scenarios. By doing this we will be capable of identify where each DBMS will be more useful. After this, we will try to understand which design decisions led to best/worst performance in a given category.

Apache Cassandra is an open source distributed database management system. Designed to handle large amounts of data across a huge number of servers, Cassandra was developed to hold the following main features: decentralized design, replication and multi data-center replication, scalability, fault tolerance, tunable consistency, Map reduce support and a support the CQL (Cassandra Query Language). MongoDB is an open source document DBMS. It's written in C/C++ . This DMBS holds the following features: Document-oriented approach, support for ad-hoc queries, replication, load balancing, file storage, aggregation, server-side java script execution, and fixed size collections. To help us study in which segment a database is better than the other we will use YCSB (Yahoo! Cloud Server Benchmark) as benchmarking tool. To do this we will first do the tests in a single node scenario, then we will do the same but with four nodes. As the resources are limited we did all this in a single machine using virtualization. We will divide our focus in analyzing the performance of each DBMS in: Writing, reading, updating, and different mixed scenarios.

II. THEORETICAL DEVELOPMENT

A. Research Question

What are the key performance differences between MongoDB and Cassandra? In what scenarios one is better than the other?

B. Dependent Variables

Runtime, throughput, operations, average latency, minimum latency and maximum latency.

C. Independent Variables

Number of nodes, machine hardware, DMBS and the benchmark scenario.

D. Hypothesis

We will not take a formal binary approach into this hypothesis testing. We will give arguments to support or deny each of these hypotheses based on the performance and scaling of each DBMS.

Null hypothesis, H0: Cassandra performance is better than MongoDB in a base scenario.

Alternative hypothesis, H1: MongoDB performance is better than Cassandra in a base scenario.

Null hypothesis, H0: Cassandra performance scales better than MongoBD.

Alternative hypothesis, H1: MongoDB performance scales better than Cassandra.

III. RESEARCH METHOD

We will run two different configurations. In a first configuration we will run the tests, only using one node. In the second configuration we will use four nodes, to test how well the single node results scale. We did all the tests using DBMS's running in Ubuntu VM, for the YCSB client we used CentOS. In all of the tests there were no more processes running than the ones from the OS and from the tested system and used default configurations. We used the default configurations of both systems.

We will run these different configurations in the two systems:

- **Workload A:** Update heavy workload. This workload has a mix of 50/50 reads and writes. An application example is a session store recording recent actions.
- **Workload B:** Read mostly workload. This workload has a 95/5 reads/write mix. Application example: photo tagging; add a tag is an update, but most operations are to read tags.
- **Workload C:** Read only. This workload is 100% read. Application example: user profile cache, where profiles are constructed elsewhere (e.g., Hadoop).
- **Workload D:** Read latest workload. In this workload, new records are inserted, and the most recently inserted records are the most popular. Application example: user status updates; people want to read the latest.

A. *Machines description*

TABLE I. VIRTUAL MACHINE HOST SPECS

Operating System	Elementary OS – 64 bits
Memory	12 GB
Processor	Intel Core i7-4710HQ 2.50GHz x 8
Graphics Card	NVIDIA Geforce GTX 850M 4GB
HDD Disc	1TB 5400 rpm

TABLE II. VIRTUAL MACHINE SPECS

Operating System	Ubuntu 14.04 LTS – 32 bits
Memory	2 GB
Processor	2 cores (of processor above)
Graphics Memory	768 MB
HDD Disc	30 GB

IV. SINGLE NODE SCENARIO RESULTS

A. *Loads*

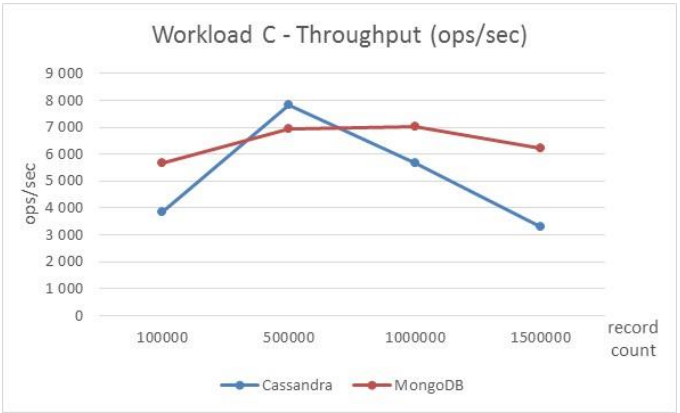

Fig. 1 - Throughput of the load operation

We chose this scenario to test the throughput of the DBMS because it's a full reading test. This way, the system has to load more data than it would in the other tests. We were able to conclude that MongoDB had a relatively stable performance in all the tests, running around 6000 operations per second. Cassandra started at 4000 operations per second, drawing a peak in Cassandra at 500 000 operations with 8000 operations per second and finishing slightly over 3000 operations per second.

B. *Tests*

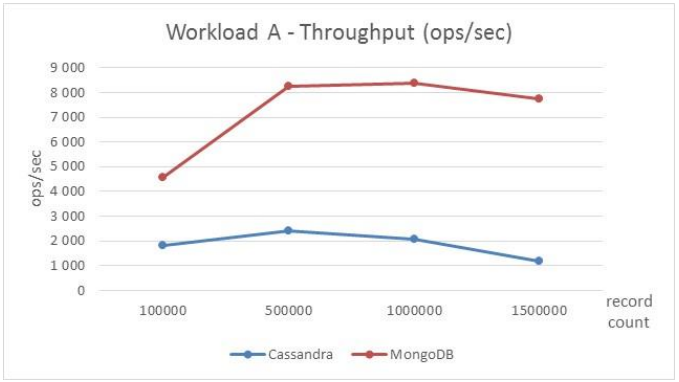

Fig. 2 – Throughput of Workload A

Workload A is an update-uheavy workload scenario, having a mix of fifty per cent reads and fifty per cent writes. This scenario simulates an application like session store, recording recent actions.

Analyzing the graphic in Fig. 2 we can see that Cassandra has its peak of performance at 500 000 operations, dropping performance after that point. MongoDB had a peak performance at 8000 operations per second, dropping performance afterwards.

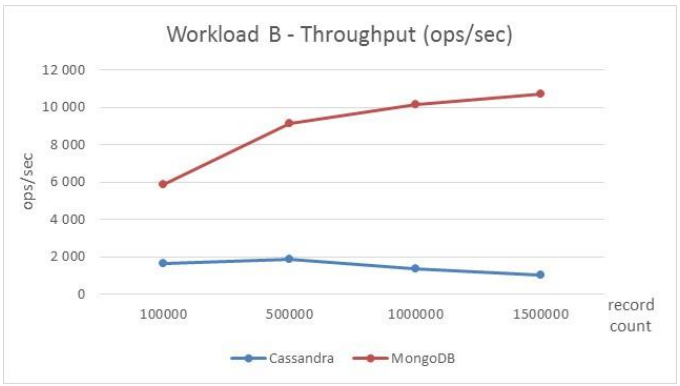

Fig. 3 – Throughput of Workload B

Workload B is a scenario made essentially of reads, and a few updates. This Workload simulates an application like photo tagging.

Cassandra reached a peak at 500 000 operations (record count), losing performance after that.

MongoDB performance continuously increased when we augmented the record count, reaching slightly bellow 11 000 operations per second.

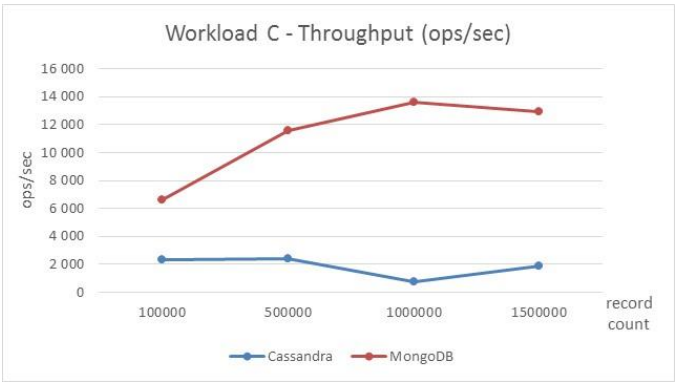

Fig 4 – Throughput of Workload C

Workload C is pure reading scenario. It simulates an application that will mostly be used to read data.

In this scenario Cassandra's performed near 2000 operations per seconds, except at the 1 000 000 operations scenario. Mongo's throughput stabilized at 12 000 operations per second, being much faster than Cassandra.

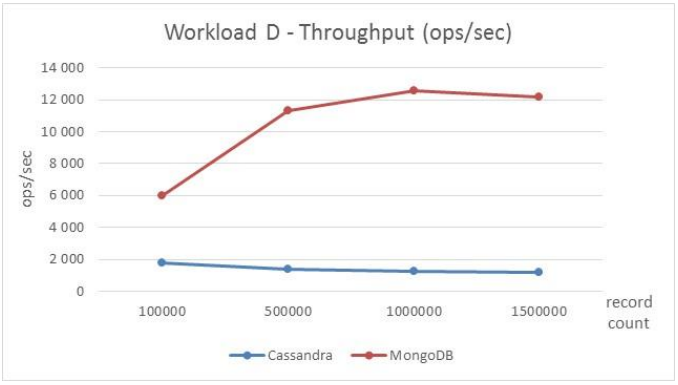

Fig 5 – Throughput of Workload D

In Workload D we had a read-latest workload scenario. Once again Cassandra was stuck near 2000 operations per second. Mongo toped at 12000 operations per second.

C. *Single Node Analysis*

Unlike we expected, MongoDB showed the best performance in every test. Cassandra seemed stuck at 2000 operations, which was really unexpected. It seems that Cassandra configurations in a single node only allowed just over 2000 operations tops.

V. FOUR-NODES CLUSTER SCENARIO RESULTS

To evaluate the scalability of the system, we did an array of tests with a four-node cluster. To deploy Cassandra's cluster we used 4 individual instances running in 4 different virtual machines. Mongo required 6 instances: 1 master, 3 slaves and 2 extra ones. The computer simply could not handle the mongo cluster, it overheated and shut itself down. So we can say that mongo requires more power to deploy a really small cluster. We did not test a "real word sized" cluster, so we really can't tell which DBMS would work better in the real world.

Due to our issues with MongoDB, we will only compare Cassandra's performance scaling between single node and four-node cluster.

A. *Loads*

In the graphic of figure 6, we have the results of the same tests we ran in Cassandra, but this time in the four-node cluster scenario.

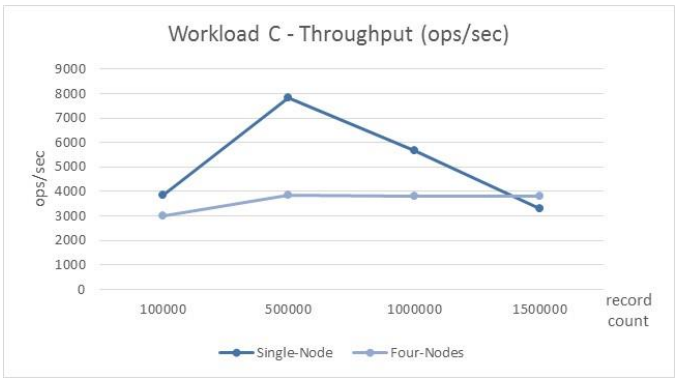

Fig 6 – Throughput of the two different scenarios (single and four-nodes).

In figure 6 we show a comparison between the loading throughputs in a single node scenario and a four-node scenario. We can clearly see that Cassandra single node behaves best in all tests except the last one. The extra overhead of the cluster slowed down the system but, in the last test, the parallelization paid off, showing better results.

B. *Tests*

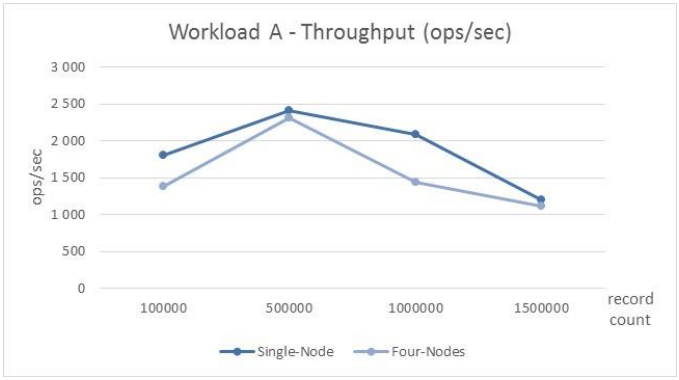

Fig 7 – Throughput workload A in the two different scenarios.

In figure 7 we have the update-heavy workload scenario. Single node scenario had better performance in all of the tests but, like in the load scenario we can see a tendency of the four-node scenario performance to become better than the single node when we increased the reading count.

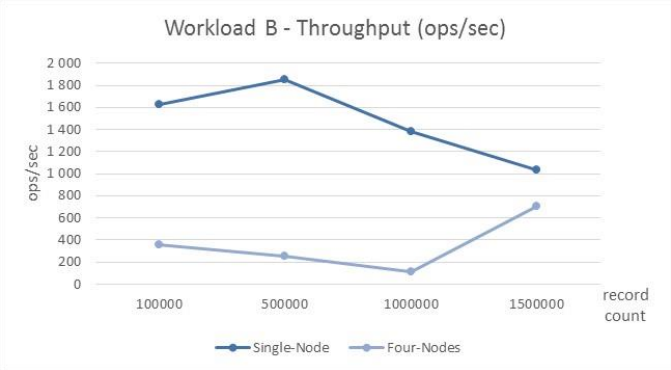

Fig 8 – Throughput of workload B in the two different scenarios.

Workload B is a read-heavy scenario, with few updates. Once again, the same thing happened. The cluster overhead begins to be meaningless when we increase the record count, beginning to seem that will behave better in a heavier scenario. Likely, we would be capable to tell this for sure if we tested for a higher record count.

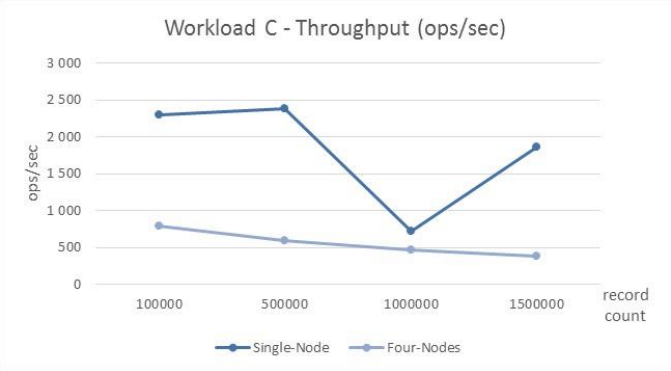

Fig 9 – Throughput of workload C in the two different scenarios.

Workload C is a read-only scenario. We can see clearly that the single node scenario behaves better than the four-nodes. Reading is Cassandra's weakest point as it's optimized for great writing throughput. Our cluster configuration didn't improve that tendency.

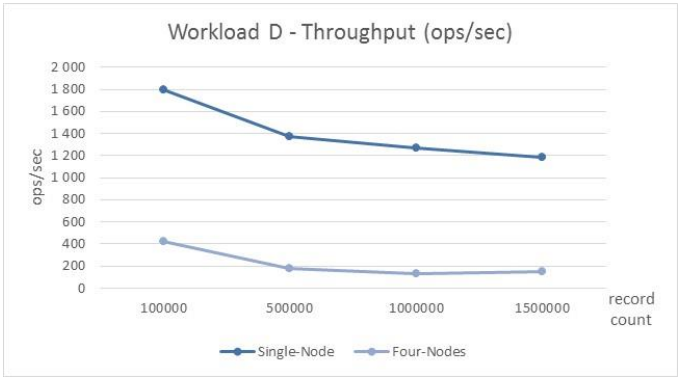

Fig 10 – Throughput of workload D in the two different scenarios.

In Workload D we had a read latest workload scenario. The curve formed by the two different scenarios was similar, but the single node scenario throughput was better. Once again this is a read scenario. Cassandra's optimizations are for writing, not reading.

VI. LIMITATIONS

As the Virtual Machines from the DEI cluster didn't provide constant processing power, we decided to use a personal laptop. This proved to be a bad decision because our machine failed to meet the minimum requirements of Cassandra and it struggled to maintain so many virtual machines.

We had only one disk, so this surely is a bottleneck. Cassandra is optimized to function in two separate disks: one to keep your CommitLogDirectory on and the other to use in DataFileDirectories.

Cassandra persists data to disk for two very different purposes. The first is to the commitlog when a new write is made so that it can be replayed after a crash or system shutdown. The second is to the data directory when thresholds are exceeded and memtables are flushed to disk as SSTables.

Commit logs receive every write made to a Cassandra node and have the potential to block client operations, but they are only ever read on node start-up. SSTable (data file) writes on the other hand occur asynchronously, but are read to satisfy client look-ups. SSTables are also periodically merged and rewritten in a process called *compaction*. Another important difference between commitlog and sstables is that commit logs are purged after the corresponding data has been flushed to disk as an SSTable, so CommitLogDirectory only holds uncommitted data while the directories in DataFileDirectories store all of the data written to a node. [2]

To test how our multiple VM setup was interfering with disk speed, we calculated the write speed and read speed of a 4.3 GB file in a single node and four nodes. Our single node setup wrote the file in 57,61 seconds, at a rate of 74.5 MB/s. The same test was performed simultaneously across the 4 Cassandra nodes, with results as follow:

Node 1: 1.1 GBs written, 47.01 seconds, 22.8 MB/s
Node 2: 1.1 GBs written, 76,13 seconds, 14.1 MB/s
Node 3: 1.1 GBs written, 67.92 seconds, 15.8 MB/s
Node 4: 1.1 GBs written, 70.78 seconds, 15.1 MB/s

As for the read speed, single node read 4.3 GBs in 63.28s at 67.9 MB/s. Across the 4 nodes, results were:

Node 1: 1.1 GBs read, 293,33 seconds, 3.7 MB/s
Node 2: 1.1 GBs read, 309.66 seconds, 3.5 MB/s
Node 3: 1.1 GBs read, 302.22 seconds, 3.6 MB/s
Node 4: 1.1 GBs read, 287.67 seconds, 3.7 MB/s

The gross difference in performance demonstrates how the bottleneck was our single hard drive disk.

We could not be sure if we had a silent process in the host machine that interfered with the results of the tests. The disk that we used had very slow speed when comparing to the ones traditionally used in clusters. Virtualization created an extra overhead.

VII. Discussion

Cassandra's lower throughput can be traced to a hard drive bottleneck. Mismatched requisites and low spec hardware contributed to below standard throughput.

Cassandra asked for 4 GB minimum RAM memory, and we only had 2 GB. Furthermore, it asked for two disks ideally. We had one shared disk between the Cassandra VM, the client and the host.

Mongo is much more less resource hungry, so this can explain why mongo beat Cassandra on all tests.

In this configuration Cassandra had a really low throughput, once again we can blame that on our hardware, which failed to meet the expected requirements. Cassandra in this setup asked for eight disks ideally, we had one and we had to share it between the four Cassandra instances, the client and the host. We think the virtualization and shared resources caused this very substandard result.

When testing MongoDB the computer overheated and shut itself down. MongoDB required six Virtual Machines to deploy six nodes, only four of which worked as Shards while the other nodes had other functions. We can state that while in single node, Mongo behaved better than Cassandra. However, it failed to scale in this virtualized scenario.

Analyzing the results we can reject the *Null hypothesis, H0:* Cassandra performance is better than MongoDB in a base scenario, but we can't really say anything about how well each system scales, because our hardware was simply not capable of handling a cluster.

VIII. Conclusion

Our main conclusion is that we can't really benchmark two DMBS in a laptop. We just can't meet the hardware requirements. Nevertheless, our best result was with MongoDB in single node scenario, simply because it demanded less requirements. In our scenario the computer simply could not handle the cluster.

Due to hardware resource limitations, we could not determine whether Cassandra would surpass MongoDB with a tailor-made host machine. Hardware scaling was therefore impossible to test.

We could have had even better throughput with MongoDB if we had installed it in an SSD disk, as it is optimized for work in SSD devices.

References

[1] Jim Gray, Database and Transaction Processing Performance, The Benchmark Handbook, 1993.

[2] CassandraHardware https://wiki.apache.org/cassandra/CassandraHardware