Cassandra's Performance and Scalability Evaluation

Melyssa Barata[1] and Jorge Bernardino[1,2]
[1]Polytechnic of Coimbra, ISEC, Rua Pedro Nunes, Quinta da Nora, 3030-190 Coimbra, Portugal
[2]Centre of Informatics and Systems, University of Coimbra, Pinhal de Marrocos, 3030-290 Coimbra, Portugal

Keywords: Performance, Scalability, NoSQL, Cassandra.

Abstract: In the past, relational databases were the most commonly used technology for storing and retrieving data, allowing easier management and retrieval of any stored information organized as a set of tables. However, today databases are larger in size and the query execution time can become very long, requiring servers with bigger capacities. The purpose of this paper is to describe and analyze the Cassandra NoSQL database using the Yahoo! Cloud Serving Benchmark in order to better understand the execution capabilities for various types of applications in environments with different amounts of stored data. The experiments with Cassandra show good scalability and performance results and how the database size and number of nodes affect it.

1 INTRODUCTION

Nowadays NoSQL databases have become the primary alternative to relational databases, with scalability, availability, and fault tolerance being key deciding factors. The environment for a NoSQL database is a largely distributed database system that allows rapid, ad-hoc organization and analysis of high-volume data types (Cattell, 2010). A flexible and schema-less data model, horizontal scalability, distributed architectures, and the use of languages and interfaces that are "not only" SQL typically characterize this technology (Moniruzzaman and Hossain, 2013); (Abramova et al., 2015).

When people discuss about performance and scalability, they very often use these two words synonymously even though they are very different (Smith and Williams, 2000). Scalability refers to the characteristics of a system to increase performance by adding additional resources. When it comes to large distributed systems, scalability is a desirable characteristic in the network, system, or process, which indicates its ability to either be prepared to grow, or handle an increasing portion of the work evenly (Pokorny, 2011). Size is just one aspect of scale that needs to be considered. Scalability may refer to various parameters of the system: how easy is it to add more storage capacity, how much additional traffic it can handle, or how many more transactions can be processed (Kuwahara et al., 2013). It provides the foundation for decisions in designing a distributed web architecture. This is an essential asset for many large-scale web applications, being able to handle extremely large amounts of users. On the other hand, performance refers to the capability of how fast something can get done (Huang and Luo, 2013). Therefore, performance is the speed at which a computer operates during a benchmark test.

Standard benchmarks are widely used for comparing the performance of different systems, answering the common question of "Which is the best system in a given domain, for specific kinds of applications?" (Barata et al., 2014). Benchmarking is an essential aspect of any database management system. In an increased competition scenario, companies are more and more faced with the need of finding management tools that allow them to diagnose critical business factors, with the purpose of doing better each time. For this reason, benchmarking exists to fit the needs of companies who seek a support tool to improve overall performance.

The main purpose of this work is to benchmark the Cassandra NoSQL database using the Yahoo! Cloud Serving Benchmark, in order to better understand their execution capabilities for various types of applications in environments with different amounts of stored data. We obtain performance and scalability results using different numbers of processing nodes that provide a better understanding on how the performance of this NoSQL is affected

Barata, M. and Bernardino, J.
Cassandra's Performance and Scalability Evaluation.
DOI: 10.5220/0005980101270134
In *Proceedings of the 5th International Conference on Data Management Technologies and Applications (DATA 2016)*, pages 127-134
ISBN: 978-989-758-193-9

by the size of the database.

The remainder of this work is organized as follows. We begin to describe in Section 2 the Cassandra database used in these experiments, and in Section 3 we explain and characterize the YCSB benchmark. Section 4 presents our experimental evaluation and lastly section 5 presents our conclusions and future work.

2 CASSANDRA

NoSQL databases were created primarily to address issues with web applications that need to operate with enormous loads of data as well as being able to scale without difficulty. Cassandra is a Column Family NoSQL database that is designed to solve the challenges associated with massive scalability. It can support a very high update throughput while delivering low latency. Cassandra is very similar to the usual relational model, made of columns and rows. The main difference is the stored data, which can be structured, semi-structured or unstructured.

When it comes to storage in clusters, all of the data is stored in distributed fashion over all nodes of the cluster. When a node is added or removed, all of its data is automatically distributed over other available nodes, and a failing node will be replaced instantly. Because of this, it is no longer necessary to calculate and assign data to each node. Cassandra's architecture is known to be peer-to-peer (partitions tasks or workloads between peers equally) and overcomes master-slave limitations such as high availability and massive scalability. Data is replicated over multiple nodes in the cluster. Failed nodes are detected by gossip protocols (peer-to-peer communication protocol in which nodes periodically exchange state information about themselves and about other nodes they know about) and those nodes can be replaced instantly (Cooper et al., 2010).

In Cassandra, data is indexed by a key that is of the type String. This key represents a line where data is found, and in each row the data is divided into columns and column families. Each column in Cassandra has a name, a value and a timestamp. Both the value and the timestamp are provided by the client application when data is inserted. Besides the normal typed columns, another kind of column exists, they are known as the super columns. The thing that differentiates these columns from the others is the fact that instead of having objects as values, they have other columns as values.

In order to group columns, Cassandra has a concept known as: Column Families, which is very similar to relational database tables. Unlike columns, the Column Families are not dynamic and must be previously declared in a configuration file. They are the unit of abstraction containing keyed rows which group together columns and super columns of highly structured data. Column families have no defined schema of column names and types supported.

Similarly to columns, there is also the Super Column Family, which is a Column family that just contains Super columns. It is useful for modeling complex data types such as addresses and other simple data structures.

Lastly, column families are grouped into Keyspaces. These Keyspaces can be compared to Schemas in a relational database.

Cassandra was designed to handle large amounts of data spread across many commodity servers. Cassandra provides high availability through a symmetric architecture that contains no single point of failure and replicates data across nodes. Cassandra's architecture is a combination of Google's Big-Table (Chang et al., 2008) and Amazon's Dynamo (DeCandia et al., 2007). It is a peer-to-peer model, which makes it tolerant against single points of failure and provides horizontal scalability. Each node exchanges information across the cluster every second. A sequentially written commit log on each node captures write activity to ensure data durability (Datastax, 2014). Data is then indexed and written to an in-memory structure called memtable, which resembles a write-back cache. Once the memory structure is full, the data is written to disk in an SSTable (sorted string table) data file (a file of key/value string pairs, sorted by keys). All writes are automatically partitioned and replicated throughout the cluster. When a read or write request is made, any node in the cluster is able to handle it. Through the key, the node that answered the requisition can know which node possesses data information.

3 YAHOO CLOUD SERVING BENCHMARK (YCSB)

The Yahoo! Cloud Serving Benchmark (YCSB) is one of the most used benchmarks, and provides benchmarking for the bases of comparison between NoSQL systems. The YCSB Client can be used to benchmark new database systems by writing a new class to implement the following methods (Cooper et al., 2010): read, insert, update, delete and scan.

These operations represent the standard CRUD operations: Create, Read, Update, and Delete.

A cloud service testing client consists of two parts: workload generator and the set of scenarios. Those scenarios are known as workloads, which perform reads, writes and updates. The YCSB predefined workloads are (GitHub, 2015):

- Workload A: Update heavy. This workload has a mix of 50% reads and 50% updates.
- Workload B: Read heavy. This workload has a 95% of reads and 5% of updates.
- Workload C: Read only. This workload is 100% read.
- Workload D: Read latest. This workload has a 95% of reads and 5% of inserts.
- Workload E: Short ranges. This workload has a 95% of scans and a 5% of inserts.
- Workload F: Read-modify-write. In this workload, the client will read a record, modify it, and write back the changes.

Due to space limits in the experimental evaluation we only show the results using the workloads: A, B, C, and D.

When a data set is generated, the client uses a set of records that define the distributions that are most likely to be chosen in the execution of operations. In this benchmark three major distributions exist (Cooper et al., 2010): Uniform, Zipfian and Latest. In our experiment we use the uniform distribution

4 CASSANDRA EXPERIMENTAL EVALUATION

In this section the experimental setup, speedup metric used and benchmarking results and analysis of execution time for the workloads used with uniform distribution will be presented.

4.1 Experimental Setup for YCSB

In this section we will describe our experiment using the YCSB benchmark for the Cassandra database. The tests were performed using three computers using the Ubuntu Operating System. The computers had the following characteristics Intel(R) Core(TM)2 Quad CPU Q8300 @2.50GHz 2.51 GHz 3,99 GB of RAM available.

For this experiment we used a one node cluster and a three node cluster, in which we loaded 10 000 000 and 100 000 000 records for the one node cluster and then for the three node cluster respectively. We performed 10 000 operations for each scenario and used 1, 100 and 1000 threads.

All executions were repeated four times, and the values presented in Section 5.2.1, 5.2.2, 5.2.3, 5.2.4, 5.2.5 and 5.2.6 are the average values of the three executions. In order to maintain all executions independent after each execution a computer restart was effected. Due to the increased speed of execution from the records in memory, this approach of testing allowed us to have isolated results whose runtimes are not influenced by the use of volatile memory or the effect of cache results.

4.2 Speedup

In this experiment we use the speedup metric. The speedup of a system is defined as the ratio between response time using a processor and using multiple processors or nodes (Karp and Flatt, 1990). The speedup measures the increase in gain in performance achieved using various processors instead of a single processor and is calculated using the following equation:

$$\text{Speedup} = \frac{\text{Response Time using } \mathbf{1} \text{ processor}}{\text{Response Time using } \mathbf{M} \text{ processors}}$$

The ratio of the execution time using 1 processor to the execution time with M processors. Ideally, one would like Speedup=M, which is called ideal speedup, although in practice this is rarely achieved. An ideal speedup means that when we increase the number of processors to M, the original response time will be reduced by the same factor.

4.3 Cassandra Evaluation

In this section the benchmarking results and analysis of execution time for workloads A, B, C, and D of uniform distribution with 10 million and 100 million records on 1 and 3 nodes will be presented.

4.3.1 Workload A

In this experiment we test workload A, which has 50% reads and 50% updates with uniform distribution. Figure 1 shows the results for 10 million records and 100 million records for a single node cluster and a 3 node cluster, with thread variation of 1,100 and 1000.

Figure 1 shows workload A which has 50% reads and 50% updates. In this figure we observe that the 3 node cluster was always faster than the 1 node cluster. This was especially noticeable when dealing with 100 and 1000 threads. We can see that the results for 100 and 1000 threads were almost the same in both scenarios. In Table 1 the speedup

results for 1 and 3 nodes are presented.

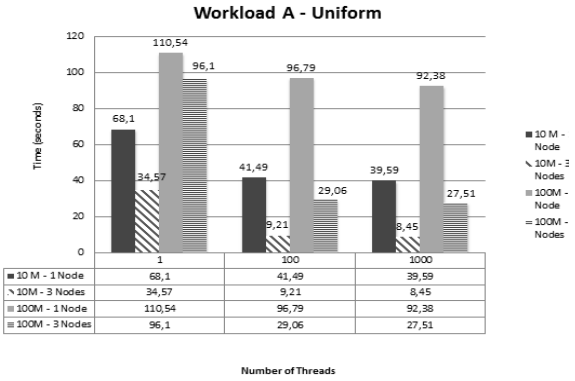

Figure 1: Execution Time of Workload A.

Table 1: Speedup results for 1 and 3 nodes (Workload A).

	1 Thread (1->3)	100 Threads (1->3)	1000 Threads (1->3)
10 M Records	1.97	4.50	4.69
100 M Records	1.15	3.33	3.36

Analyzing Table 1 we conclude that in the 100 and 1000 thread case for the 3 node cluster in comparison to the 1 node cluster the performance results were always superlinear. With 100 threads we obtained speedup results that were 4.5 times faster for 10 million records and 3.33 times faster for 100 million records. For the 1000 thread case scenario we see that it was 4.69 and 3.36 times faster when using the 3 node cluster in comparison to the 1 node cluster in both 10 million and 100 million records respectively. In both these cases the reason for these superlinear results is due to the fact that with one node there is always a limit, but if we increase the number of nodes to 3 we are incrementing power to a multiple factor of 3. The overall gain is limited by the processing power of the node which is not able to process all of the 100 threads at the same time, however, when we increase the cluster size to 3 we are dividing more data and because of this more gain is achieved. What this means is that we are getting rid of this limit that exists with the single cluster because we increase processing power to 3 and therefore 300 threads are going to be working simultaneously for the 3 node cluster. However, when analyzing 1 thread we observe that a speedup result of only 1.97 was obtained for 10 million records, and a speedup result of 1.15 for 100 million records. The reason why the gain here was sublinear is because we increased computing power by 3 and the additional node overhead (any excess computation time, memory, bandwidth or other resources) of network communication is greater than this gain.

In conclusion, we expected some factors from this experiment to have been different. For instance, the performance was expected to improve approximately 3 times more from the single node cluster to the 3 node cluster in all threads. However in the 1 thread scenario this did not happen because in the 3 node cluster the database had to transfer more information by network to execute each query. Also, there was no improvement in execution time from 100 to 1000 threads because of the network overheads and the processing overheads. This could be because the processor may not support more threads at the same time due to the fact that both the memory and the processor have a limit. Because of this, it is apparent that from 1 thread to 100 threads saturation was achieved. Since no improvements were calculated from 100 to 1000 threads, it becomes evident that at 100 threads our experiment reached the limit of hardware resources available. Table 2 presents the percentage gain obtained for workload A.

Table 2: Percentage Gain for Workload A.

	1 –> 100 Threads	100 -> 1000 Threads
10 M Records – 1 Node	39%	5%
10 M Records – 3 Nodes	73%	8%
100 M Records – 1 Node	12%	5%
100 M Records – 3 Nodes	70%	5%

In terms of gain, we conclude from Table 2 that better results were achieved when going from 1 to 100 threads in all four cases. Distinct values between different record sizes and nodes were noticed with 100 threads especially when dealing with 3 nodes. In both 10 and 100 million records with 3 nodes there was a gain of 73% and 70% respectively. These results were undoubtedly better in comparison to using a single node. When using 10 million records and 100 million records when going from 1 to 100 threads with the single node we see a percentage gain of 39% for 10 million records, and only a 12% gain with 100 million records. On the other hand, when going from 100 threads to 1000 threads we see that almost no gain was achieved. Percentage gain in this case when dealing with 10 million records and 3 nodes was 8%, and only 5% when dealing with 100 million records with 1 and 3 nodes, and 10 million records with 1 node.

4.3.2 Workload B

In this experiment workload B was tested with

uniform distribution. This workload has 95% reads and 5% updates. Figure 2 shows the results for 10 and 100 million records for a 1 and 3 node cluster, with 3 different thread variations (1, 100 and 1000).

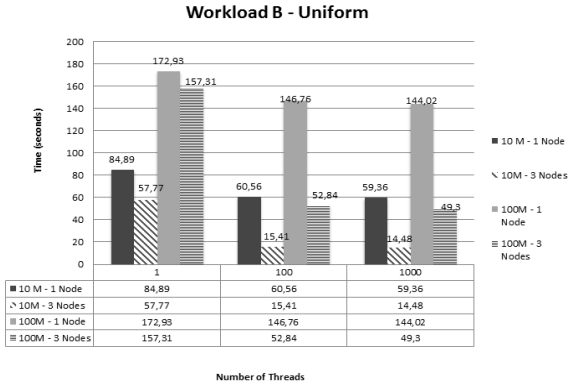

Figure 2: Execution Time of Workload B.

Figure 2 shows workload B, from this figure we conclude that that 3 node cluster always had faster execution times than the 1 node cluster. In Table 3 we present the speedup results obtained for this workload.

Table 3: Speedup results for 1 and 3 nodes (Workload B).

	1 Thread (1->3)	100 Threads (1->3)	1000 Threads (1->3)
10 M Records	1.47	3.93	4.10
100 M Records	1.10	2.78	2.92

In the 100 and 1000 thread case the performance results obtained when using 10 million records from 1 node to 3 nodes were superlinear in both cases. For 100 threads it was 3.93 times faster and with 1000 threads it was 4.10 times faster as can be seen from Table 3. With 100 million records the results we obtained were also good having achieved an almost linear speedup, 2.78 for 100 threads and 2.92 for 1000 threads. When using 1 thread the results obtained were sublinear for both 10 million and 100 million records. When using 10 million records we obtained a result of 1.47 from 1 node to 3 nodes, and a result of 1.10 was calculated when using 100 million records. In both 100 and 1000 threads the results were very similar, varying only by a few seconds from 10 million records to 100 million records as we can see from Figure 8. This is because various threads are going through the CPU, and the CPU can only do so many things at a given time, therefore if a certain threshold is hit, it doesn't matter how many more things are trying to get through the CPU, it is still limited by what the CPU can process at a given time. Just because we add more threads, does not mean we will obtain better performance results. This workload in general achieved slower results than those presented in workload A because Cassandra is optimized for write-heavy workloads. In Table 4 we present workload B's percentage gain.

Table 4: Percentage Gain for Workload B.

	1 -> 100 Threads	100 -> 1000 Threads
10 M Records – 1 Node	28%	2%
10 M Records – 3 Nodes	73%	6%
100 M Records – 1 Node	15%	2%
100 M Records – 3 Nodes	66%	7%

When comparing the results obtained from Table 4 we can see that 100 and 1000 threads had very different behavior. From 1 to 100 threads the 3 node cluster once again had better results in both 10 and 100 million records with 73% and 66% percentage gain respectively, while the single node cluster had a gain of 28% for 10 million nodes, and 15% for 100 million nodes. However, when comparing the results obtained from 100 to 1000 threads we conclude that the gain was little to none in this experiment. The 3 node cluster here once again had a bit more gain than the 1 node cluster with 6% for 10 million records, and 7% for 100 million records, while the single node cluster had a gain of only 2% for both 10 and 100 million records.

4.3.3 Workload C

In this experiment we test workload C which has 100% reads with uniform distribution. Figure 3 shows the results for 10 million records and 100 million records for a single node cluster and a 3 node cluster, with thread variation of 1, 100 and 1000.

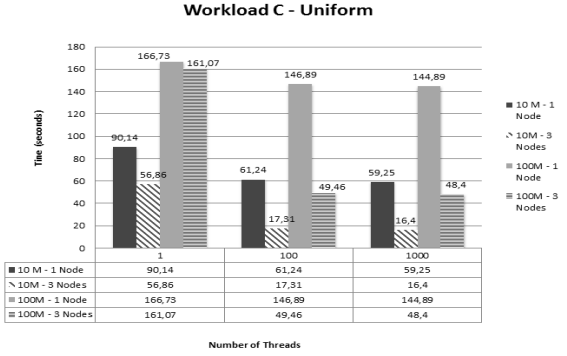

Figure 3: Execution Time of Workload C.

In Figure 3 workload C results are presented. For

this workload we have 100% of reads. The results obtained from this experiment were similar to those obtained from workload B which had 95% reads and 5% updates. When comparing execution times, we can see a major improvement in time for 100 and 1000 threads in both 10 million and 100 million records from 1 to 3 nodes. In Table 5 the speedup results for 1 and 3 nodes are presented.

Table 5: Speedup results for 1 and 3 nodes (Workload C).

	1 Thread (1->3)	100 Threads (1->3)	1000 Threads (1->3)
10 M Records	1.59	3.54	3.61
100 M Records	1.04	2.97	2.99

From Table 5 we see that for 100 threads from 1 to 3 nodes it is 3.54 times faster with 10 million records and 2.97 times faster with 100 million records. On the other hand with 1000 threads comparing the 1 node cluster to the 3 node cluster we conclude that with 10 million records we obtained an improvement of 3.61 and with 100 million records we got and improvement in speedup of 2.99. Once again the same scenario from the two previous workloads was observed with 1 thread. With 10 million records the execution time was 1.59 times faster comparing 1 node to 3 nodes and with 100 million records the result was 1.04 times faster with the 3 node cluster. These results are sublinear, and have to do with the fact that the database had to transfer more information by network to execute each query and because computing power was increased to 3, the node overhead of network communication proved to be more than the gain we obtained. As mentioned before, Cassandra is optimized for write-heavy workloads, because of this Cassandra's efficient sequential use of disk for updates reduces contention for the disk head. However, the results here for both 100 and 1000 threads were as expected in terms of scalability because as the number of nodes increased from 1 to 3, the results obtained also improved 3 times more seeing as the execution time decreased significantly from 1 to 3 nodes. Table 6 presents the gain in percentage we calculated for workload C.

Table 6: Percentage Gain for Workload C.

	1 -> 100 Threads	100 -> 1000 Threads
10 M Records – 1 Node	32%	3%
10 M Records – 3 Nodes	70%	5%
100 M Records – 1 Node	12%	1%
100 M Records – 3 Nodes	69%	2%

From Table 6 we conclude that from 1 to 100 threads the results obtained were better than from 100 to 1000 threads. With 100 threads we see that for the 3 node cluster we had better percentage gain in both 10 and 100 million records with 70% for 10 million records and 69% for 100 million records. In case of the single cluster we see a gain of 32% for 10 million records, and only 12% for 100 million records. However, for 1000 threads the gain was once again close to none. For 10 million records a gain of 3% was calculated in the single node cluster, and 5% in the 3 node cluster. For 100 million records a gain of only 1% for the 1 node cluster was obtained and 2% for the 3 node cluster.

4.3.4 Workload D

In this experiment we test workload D which has 5% inserts and 95% reads with uniform distribution. Figure 4 shows the results for 10 million and 100 million records for a single node cluster and a 3 node cluster, with thread variation of 1, 100 and 1000.

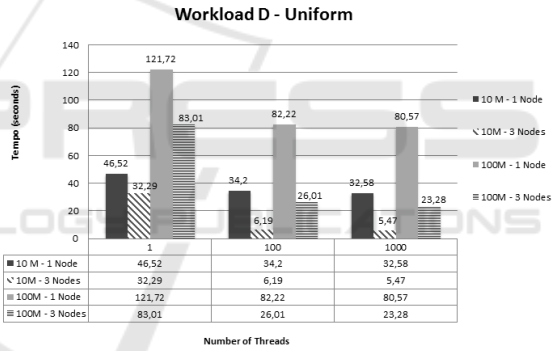

Figure 4: Execution Time of Workload D.

Figure 4 shows workload D which includes 5% of insert and 95% of read operations. This workload is similar to workload B which has 95% reads and 5% updates. In terms of execution times even though both workload B and D contain 95% reads, the insert and update part of the workload changed everything seeing as workload D presented faster results than workload B. This is because in workload B an update needs to firstly scan the whole table in order to get the records to update, and in workload D data is just simply inserted. For workload D the results displayed major differences when comparing 1 thread to 100 and 1000 threads. Table 7 shows the speedup results for the single node and 3 node cluster.

Table 7: Speedup results for 1 and 3 nodes (Workload D).

	1 Thread (1->3)	100 Threads (1->3)	1000 Threads (1->3)
10 M Records	1.44	5.53	5.96
100 M Records	1.47	3.16	3.46

From Table 7 we see that the 1 thread scenario has an improvement in terms of speedup that is 1.44 times faster with 10 million records and 1.47 times faster with 100 million records when using the 3 node cluster as opposed to the 1 node cluster. The results obtained here were not as expected in terms of scalability because as the number of nodes increased from 1 to 3, the results obtained were also supposed to improve nearly 3 times more. Just like in the previous workloads the explanation here is the same seeing as additional node overhead of network communication was greater than the gain. However, when comparing the results for 100 and 1000 threads we see an improvement in execution time in both threads. The results here were superlinear. With 100 threads we obtained a speedup result of 5.53 with 10 million records and 3.16 with 100 million records. In the 1000 thread case we obtained a speedup results that was 5.96 times faster with 10 million records and 3.46 times faster with 100 million records from 1 node to 3 nodes. In these four cases the time reduced by 3 from 1 node to 3 nodes, making these results super, seeing that as we increased the number of nodes from 1 to 3 the results also improved 3 times more. However, seeing as the results from 100 to 1000 threads were very similar in terms of execution time, saturation was achieved at 100 threads. Table 8 presents the percentage gain results for workload D.

Table 8: Percentage Gain for Workload D.

	1 -> 100 Threads	100 -> 1000 Threads
10 M Records – 1 Node	26%	5%
10 M Records – 3 Nodes	81%	12%
100 M Records – 1 Node	32%	2%
100 M Records – 3 Nodes	69%	10%

After analyzing Table 8 we conclude that the results here were much better when going from 1 to 100 threads as opposed to 100 to 1000 threads. 100 threads had superlinear results with 3 nodes having a gain of 81% for 10 million records and 69% for 100 million records. For the single node cluster we obtained a gain of 26% for 10 million records, and 32% for 100 million records. With 1000 threads we see that better results were calculated once again with the 3 node cluster in comparison to the 1 node cluster. Having the 3 node cluster a gain of 12% for 10 million records, and 10% gain for 100 million records. For the single cluster only a 5% gain was obtained for 10 million records, and 2% for 100 million records.

5 CONCLUSIONS AND FUTURE WORK

The popularity of NoSQL databases has increased in recent years because they bring a number of advantages compared to relational databases. In our work we experimentally evaluated Cassandra, one of the most popular Column family NoSQL databases.

Throughout the experimental evaluation we assessed performance and scalability of the Cassandra database using the Yahoo! Cloud Serving Benchmark. The workloads that were used were defined in a range of scenarios. We tested factors such as data size, number of nodes, number of threads, workload characteristics, and analyzed whether desirable speedup and scalability properties were met. By analyzing the results we concluded that Cassandra exhibits good scaling capacity while maintaining the performance which leads us to say this database is highly optimized to work with large volumes of data. After concluding our results we expected some factors from this experiment to have been different. For instance, the performance was expected to improve 3 times more from the single node cluster to the 3 node cluster in all thread cases. However in the 1 thread scenario this did not happen in any workload because in the 3 node cluster the database had to transfer more information by network to execute each query making the gain sublinear because we increased computing power by three and the additional node overhead of network communication is greater than this gain. Also, there was no major improvement or difference in execution time when going from 100 to 1000 threads because of the network overheads and the processing overheads. This could be because the processor may not support more threads at the same time due to the fact that both the memory and the processor have a limit. Since various threads are going through the CPU, and the CPU can only do so many things at a given time, if a certain threshold is hit, it doesn't matter how many more things are trying to get through the CPU, it is still limited by what the CPU can process at a given time. Just because we add more threads, does not mean we will obtain better performance results. Because of this, it is apparent that from 1 thread to 100 threads saturation was achieved. In terms of gain, we can also conclude that

far better results were achieved when going from 1 to 100 threads when using 10 and 100 million records for 1 and for 3 nodes. Distinct values between different record sizes and nodes were noticed when going from 1 to 100 threads especially when dealing with 3 nodes. In both 10 and 100 million records with 3 nodes there was almost always a better gain in comparison to using a single node.

In general Cassandra's workload executions are fast except when it is necessary to execute scans, in those cases the performance highly decreases and the system is not as fast. It is important to remember that regardless operation type the system must be scaled adequately, according to the database size.

As future work, we intend to analyze and compare the variation of execution time, performance and scalability between different types of NoSQL databases using the YCSB benchmark by increasing record size, using more clusters and using more operations. This approach would enable us to better understand how NoSQL and relational databases differ from one another comparing which one is better for different purposes.

REFERENCES

Abramova V., Bernardino J., Furtado P. (2015), SQL or NoSQL? Performance and scalability evaluation. *IJBPIM 7(4)*: 314-321 (2015)

Barata, M., Bernardino J., and Furtado P. (2014), "YCSB and TPC-H: Big Data and Decision Support Benchmarks," *2014 IEEE International Congress on Big Data*, Anchorage, AK, 2014, pp. 800-801.

Cattell R. (2010) "Scalable SQL and NoSQL data stores" *SIGMOD Record* Vol. 39 No. 4 pp. 12-27.

Chang F., Dean J., Ghemawat S., Hsieh W., Wallach A., Burrows M., Chandra T., Fikes A., and Gruber R. (2008) "Bigtable: A distributed storage system for structured data" *ACM Trans. Comput. Syst.* (26) no. 2.

Charsyam - Cassandra Data Model - https://charsyam.wordpress.com/tag/cassandra-data-model/ Accessed 08-01-2015.

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R. (2010) "Benchmarking Cloud Serving Systems with YCSB", *SoCC* pp. 143-154.

Data Magnum –Graph Databases (Including Object DBS) - http://data-magnum.com/lesson-8-graph-databases-including-object-dbs/ Accessed 29-11-2015.

Datastax - Apache Cassandra™ 1.1 Documentation - http://www.datastax.com/doc-source/pdf/cassandra11.pdf - Accessed 31-05-2014.

DeCandia G., Hastorun D., Jampani M., Kakulapati G., Lakshman A, Pilchin A, Vosshall P., and Vogels W., (2007) "Dynamo: Amazon's Highly Available Key-Value Store", *SOSP* pp.205-220.

GitHub - YCSB. - https://github.com/brianfrankcooper/YCSB/wiki/core-workloads - Accessed 02-11-2015.

High Scalability - 5 Steps To Benchmarking Managed NoSQL - DynamoDB Vs Cassandra - http://highscalability.com/blog/2013/4/3/5-steps-to-benchmarking-managed-nosql-dynamodb-vs-cassandra.html - Accessed 04-11-2015.

Huang, Y. and Luo T. (2013) "NoSQL Database: A Scalable, Availability, High Performance Storage for Big Data". ICPCA/SWS pp. 172-182.

Karp, A. and Flatt H. (1990) "Measuring Parallel Processor Performance", Commum. ACM Vol. 33 No. 5, pp.539-543.

Kuwahara, H., Fan, M., Wang, S., Gao, X., (2013) "A framework for scalable parameter estimation of gene circuit models using structural information". Bioinformatics, Vol. 29 No. 13 pp. 98-107.

Lakshman, A. and Malik P. (2010) "Cassandra – A Decentralized Structured Storage System", Operating Systems Review Vol. 44 No. 2 pp. 35-40.

MongoDB – MongoDB CRUD Introduction. http://docs.mongodb.org/manual/core/crud-introduction/ Accessed 29-11-2014.

Moniruzzaman A. B. M. and Hossain S. A. (2013) "NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison." *CoRR* Vol. abs/1307.0191.

Pokorny J. (2011) "NoSQL databases: a step to database scalability in web environment." pp. 278-283.

Smith C., and Williams L. G. (2000) "Performance and Scalability of Distributed Software Architectures: An SPE Approach". *Scalable Computing: Practice and Experience* Vol. 3 No. 4.