Comparison of database and workload types performance in Cloud environments

George Seriatos , George Kousiouris, Andreas Menychtas, Dimosthenis Kyriazis, and Theodora Varvarigou,

National Technical University of Athens, 9 Heroon Polytechniou Str,
15779 Zografou, Greece
{g.seriatos,gkousiou, ameny, dimos, dora}@mail.ntua.gr

Abstract. The rapid growth of unstructured data over the last few years, has led to the emergence of new database management systems. Traditional relational databases, despite their wide adoption and plethora of features, begin to show weaknesses when having to deal with very large amounts of data. Numerous types of databases have emerged in the Cloud domain, in order to exploit the elasticity of Cloud environments, while relaxing the typical ACID considerations and investigating trade-offs of the CAP theorem. The aim of this paper is to investigate how such offerings (MongoDB, Cassandra and HBase namely), based on these tradeoffs, behave when deployed in virtual environments (of the BONFIRE facility) and how they are measured against widely used benchmarks such as YCSB. The results may be helpful for potential adopters to choose from these offerings, based on their individual needs for specific workloads or query structures.

Keywords: Cloud computing, NoSQL, Performance, YCSB, HBase, MongoDB, Cassandra, benchmarks

1 Introduction

In recent years, the generation and storage of enormous data sizes has led to the creation and investigation of numerous, tailored per case data management systems, that go beyond the typical SQL databases applications. It is estimated that the total data size of the digital universe is equivalent to 2.84 zettabytes (billion terabytes) with predictions raising this number to 40ZB for 2020[1]. Facebook alone gathers 300 petabytes of data, from which it processes at least 1 PB per month[2]. The Large Hadron Colider in CERN gathers approximately 15 petabytes per year for processing[3]. The term Big Data is used for one of the most emerging technologies in order to describe the concentration, storage and analysis of especially large data volumes for the extraction of conclusions, correlations and trends. Areas that are affected by this analysis include meteorology, genomics, market analysis among others.

Data management solutions can considerably benefit from their deployment and instantiation in cloud computing environments [16], however their performance may often differ, significantly in many cases depending on the configuration and offering, which affects the operational aspects of each solution so as to effectively exploit the cloud computing innovations [17]. However, the operation in an abstracted and distributed environment introduces also significant challenges e.g. the CAP theorem [18] which states that you can obtain at most two out of three properties: Consistency, Availability and tolerance to network Partitions. In large scale data management approaches where data are replicated and distributed, consistency is compromised in order provide high availability, thus relaxing the ACID guarantees (Atomicity, Consistency, Isolation, Durability) of the system for database transactions [19]

Traditional relational databases portray a number of significant weaknesses in the analysis of these data, that may origin either from their sheer volume or from the fact that in many cases they follow unstructured data formats that are difficult to be translated into rigid database schemas. While the SQL solutions are oriented towards aspects such as consistency and concurrent transactions, they fall short in use cases where large partitioning or availability needs are necessary. This gap has started to be covered in recent years through the development of NoSQL systems that tend to abandon some of the typical characteristics of relational databases (such as ACID characteristics) in order to ensure the ability for parallel and distributed storage and processing without structural constraints (e.g. table structures).

Development of NoSQL systems has been extremely rapid in recent years. Currently there are 150 available systems[4]. The separation in categories is performed mainly through their capabilities or through the way they store data. An indicative categorization of NoSQL systems appears in [5]. The main purpose of this paper is to investigate a number of such solutions (namely MongoDB, HBase and Cassandra) in a variety of usage scenarios, based on different types of workloads, and extract a number of measurements and conclusions with relation to each system's ability to handle the respective traffic. To this end, the YCSB benchmark client[6] is used in order to launch queries against deployed system instances in the Bonfire experimental Cloud platform[7]. The paper is structured as follows. In Section 2 related work in the respective field of Cloud and DB benchmarking is presented, while in Section 3 the key characteristics of the selected databases are presented, along with information on the automation and setup of the measurement process. Section 4 presents the performed experiments and measurements results while Section 5 concludes the paper.

2 Related Work

. There are several approaches that analyze the performance aspects of the database management solutions in cloud. In RDBMS, the TPC benchmarks were the most prominent benchmarking approaches for measuring the performance characteristics [10]. [11] conducted benchmarks on several existing cloudbased management systems: a) data read and write benchmark with seven tasks to evaluate the read and write performance in different situations, and b) structured query benchmark focusing

on basic operations in the structured query language such as key words matching, range query and aggregation.

NoSQL cloud management systems can be categorized as key-value stores, document stores, column stores and graph stores[12]. Their characteristics include a variety of different data models (fully structured semi structured and unstructured), different querying and most importantly different scaling methodologies to support data partitioning, replication, consistency and concurrent access. These characteristics, the execution environment configuration, have immense effect on the performance of the data management operations, positive or negative. Brian F. Cooper et al. propose YCSB framework [6] that facilitates performance comparisons of modern cloud data serving systems and define a core set of benchmarks and report results for four widely used systems: Cassandra, HBase, Yahoo!'s PNUTS and MySQL. Their analysis examines the aspects of a) read performance versus write performance, b) latency versus durability, c) synchronous versus asynchronous replication and d) data partitioning, in operation of read, update, scan and insert.

Authors in [13] argue that standardized performance benchmarking is required so as to evaluate the eventual consistency in distributed key-value storage systems and propose a methodology that extends the popular YCSB benchmark to measure the staleness of data returned by reads using the concept of Δ-atomicity [14]. [15] presents an evaluation of the performance for database management, SQL and NoSQL, in the domain of IoT and particularly for sensor data. Besides the difference in performance between SQL and NoSQL solutions, their analysis results show a considerable impact on the performance when the databases are deployed in virtualized cloud environments. In most cases the impact is negative however, only a specific deployment has been tested and no other cloud offerings and/or configurations are examined.

3 DB Features and Measurement Automation

3.1 DB features

With regard to the selected databases, the goal was to differentiate between features and strategies of available systems. Thus in terms of architecture, HBase follows a more centralized master slave approach, while Cassandra a more peer to peer one. HBase is written in Java and is tightly integrated with the underlying file system (HDFS) and with the MapReduce Apache Hadoop framework and can offer consistency guarantees. It follows a key value approach and stores data in a column oriented format on disk. It also offers atomicity of operations on a row level. Cassandra is also written in Java and the main difference is that it does not portray a single point of failure. In order to enroll a node in the system, one only needs to start the basic daemon process and insert information on one existing system node. In practice a number of nodes are determined as seeds and they are the ones that undertake the role of enrolling a new node in the ring. Cassandra's main benefit is the lack of one master node, which improves system resiliency and availability, however

it comes with a price on the consistency levels achieved. It follows a column oriented data organization. One extra feature with relation to HBase is the ability to define composite column families that can serve as an extra layer of organization. Thus it may depict concentrated data from multiple columns. Cassandra also offers atomicity on a row level, however contrary to HBase it can not offer atomicity in cases of updating more than one rows in a single transaction. Different consistency levels are offered, that are coupled with the used replication factor and they regulate the need to have consistent replicas across the system.

MongoDB on the other hand is a document oriented DB, meaning that it stores the data in fields and can be directly queried based on their contents. Data are stored in the form of BSON (Binary JSON). It may also support secondary indexes for faster search. The replica sets in Mongo are defined as primary or secondary. Update of the secondary instances is performed synchronously, affecting latency, by adjusting the "write concern" option or asynchronously, in order not to bottleneck the system. However the latter has an effect on consistency for read operations from them. Based on the consistency option selection this read operation may be limited only to the primary copy. A major difference of this DB is the fact that it mainly uses memory-mapped files in order to enhance performance.

3.2 Cloud Facility Setup

The experiments were performed in the BonFIRE Cloud computing testbed[7]. The purpose of this facility is to provide an experimental facility for Cloud Computing research, across various locations and heterogeneous resources. Management of the available resources is performed through OCCI[8], in order to offer a homogeneous interface with heterogeneous infrastructures (OpenNebula, Virtual Wall, Cells). For the purposes of the experiment, the resources in Table 1were used. Node 1 was selected with increased capabilities in order to serve as the Master node in HBase and HDFS and it was enriched with more functionalities in the other two systems as well. OS in all cases was Debian Squeeze v6 (kernel:2.6.32-5-amd64).

Table 1: BonFIRE resources used

	Node 1	Nodes 2-6	Node 7
CPU	4 Physical Cores (AMD Opteron 6176)	2 Physical Cores (2:AMD Opteron 6176, 3-6: Intel Xeon E5620)	4 Physical Cores (AMD Opteron 6176)
RAM	10 GB	4 GB	1 GB
Disk	10GB ext3 : OS + Software 10GB ext4 : Data	10GB ext3 : OS + Software 10GB ext4 : Data	10GB ext3 : OS + Software
Software	HBase 0.94.17 Hadoop 1.2.1 Zookeeper 3.4.5 Mongo 2.4.10 Cassandra 2.05	HBase 0.94.17 Hadoop 1.2.1 Zookeeper 3.4.5 Mongo 2.4.10 Cassandra 2.05	YCSB
Functionality	HBase Datanode, Namenode, Secondary Namenode, RegionServer, Zookeeper, Cassandra Daemon, Mongod, Mongo Router, Config Server	HBase Datanode, Cassandra peers, Mongod, Mongo Router (Nodes 2-4)	YCSB client

HBase was the system that presented the most challenges in terms of setup, since in many cases the errors occurred were not adequately described. Special care was given to the networking setup (especially to restrict the use of IPv6), and to remove the loopback address since it caused connection errors to the other nodes. It was also the system that appeared to be more affected by the RAM shortage in the available nodes.
Cassandra was the system that was easier to manage and configure, through the configuration of seed node details. In the case of Mongo, a series of manual steps were necessary. In order to ensure a replication factor of 3, 3 mongod processes need to start, that are configured with relation to which one is the primary, in order to kick off the different shards.
The distribution of the replicas followed the logic of a Cassandra ring. Ext4 file system was used for the main data since it is considered more efficient. One characteristic of Mongo is that due to the preallocation techniques used, it requires significantly higher disk space to start, in comparison to the actual data stored. Given that the amount of disk space was limited, a number of options needed to be utilized during the configuration that limit the initial size of the DB ("--smallfiles" and "--oplogSize 128").
Due to the fact that the existence of replication indirectly reduces disk space for the original data, the limit of entries to the DB was set to 1.8 Million, given the available resources. Each record consisted of 10 fields and each field of 100 bytes, resulting in 1KB per record. Thus the overall actual data used in the tested systems were around 5.4 GB.

3.3 Measurement process and execution automation

The actual benchmark execution is performed through the use of YCSB, a client that is responsible for creating queries against the target databases, based on the input parameters that define the type of operations, and for connecting and submitting the queries based on a set of drivers for each system. One key parameter that needs to be clarified is the Throughput (in Ops/sec) which is the desirable number of operations that must be achieved by the system. This does not necessarily mean that the system will achieve this rate however. Latency of the respective operations is also logged and monitored. Average values for these metrics are reported in the end of the measurement cycle. YCSB also contains a set of default workloads that are indicative of specific use cases. Examples of this are Workload A (typical of user session storing for action logging), Workload B (photo tagging in social networks), Workload C (caching of data), Workload D (user status in social networks), Workload E (forum discussions retrieval) and Workload F (user management DBs). More details on these workloads are given in Section 4.

The automated framework for the experiment execution appears in Figure 1. The user inputs a number of parameters such as the number of nodes, the DB size in terms of records, desired throughput, what type of DB to setup etc. This information is then passed to the partial executors, which are responsible for creating the resources on BonFIRE (via OCCI), configuring the nodes via ssh based on the DB type (setting up

seed files, alerting where to find the Namenode, creating, populating or cleaning up tables etc.) and finally launching the YCSB client to perform the queries.

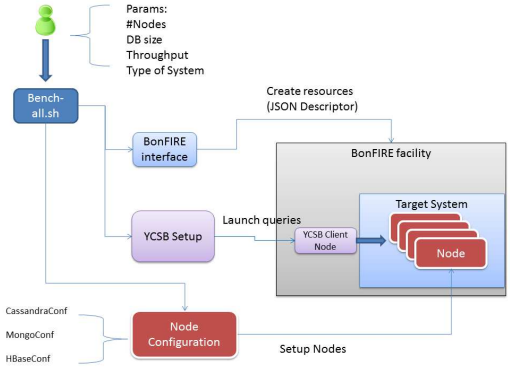


Figure 1: Automated execution framework

4 Experiments and Results

In order to perform the experiments against the deployed databases, the YCSB benchmarking client was used. YCSB produces queries against the former, based on a throughput that is determined by the parameters of execution. In reality, while this throughput is set, it is perceived as the target limit. However it is limited by the size and endurance of the underlying system. Thus, while the set throughput was starting from 1000 operations per second and increased each time by a thousand, the actual achieved rates were not completely aligned, as will be seen by the measurements. Each measurement (for a given system, workload and set throughput) was performed 4 times and the average was calculated. In many cases there were deviations that can be attributed to the operation of the underlying Cloud service. Timing constraints were also used, meaning that each series of measurements included an overall maximum time for completion. This maximum time was calculated based on the types of actions performed against the DB and the necessary throughput. If that throughput was less than the 2/3 of the needed limit, then the experiment was stopped since it would not add any additional information for the charts and would introduce unnecessary delays. Following, the results per type of workload are presented.

4.1 YCSB Workload A (50% updates-50% reads)

In workload A, HBase and Cassandra have achieved a significantly higher throughput than MongoDB, as it appears in Figure 2. The reason for this is the

increased updates ratio. MongoDB returns an update success when this is registered in RAM, providing relatively relaxed persistency guarantees, thus this element is not portrayed in the latency. Persistency of data is programmed every 100 ms through the journaling mechanism and the synchronization between the RAM data and disk file data is performed every 60 seconds. However, due to the limited memory of the used system, it appears that the OS was synchronizing the files in shorter intervals in order to free memory space and fetch the necessary files for the read operations.

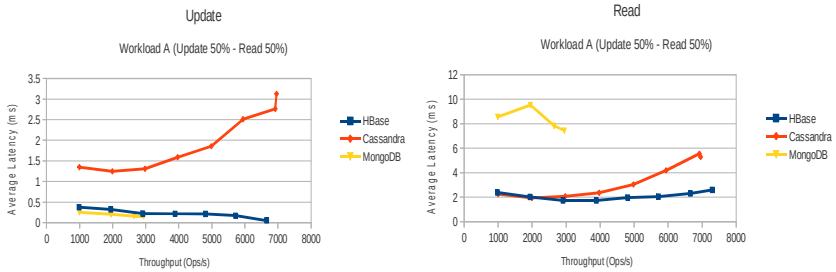
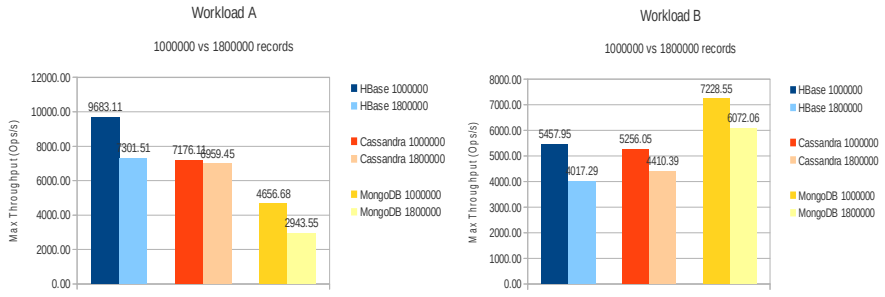


Figure 2: Comparative Latency vs Throughput in YCSB Workload A

On the contrary, HBase and Cassandra are not affected by updates since these approaches do not fetch the data to be updated from the disk bit perform the changes in new files (HFiles and SSTables) by using disk serial write capabilities. The update of the data happens in the background by merging these files, without influencing significantly the performance of these systems due to the retrieval times of files from disk. With regard to the effect of data volume in DB performance (Figure 3), MongoDB seems to be significantly affected (for the same reasons mentioned above) and its performance is reduced by 37% when operations are increased to 1.8million, in comparison to 1 million. When data are of a smaller volume, caching mechanisms portray a larger hit ration and databases perform significantly better. HBase also dropped by 25%, showing the need for more RAM. Cassandra on the other hand was not significantly affected, portraying a deterioration of 3%.

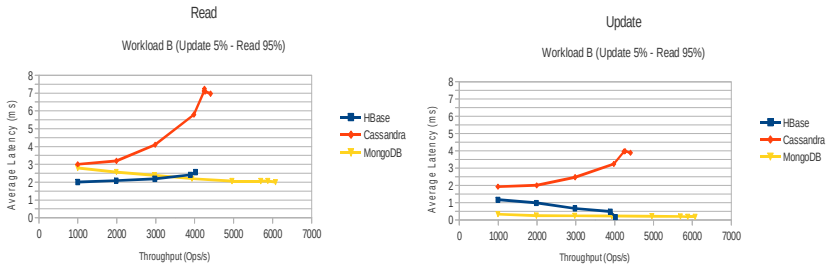


(a) (b)

Figure 3: Throughput Comparison for different DB sizes in a) Workload A and b) workload B

4.2 YCSB Workload B (5% updates-95% reads)

In workload B, MongoDB performed significantly better (Figure 4) . The memory mapped files in conjunction with the Zipfian distribution of YCSB, that selects a subset of data to perform the multitude of operations, have enabled the caching mechanisms to be exploited. Lack of increased updates has also helped towards this direction. HBase and Cassandra performed significantly worse than in workload A, due to the limited memory dedication to caching, since this value is a percentage of the Heap size (default 1GB) used in every node. On the other hand, MongoDB allocated dynamically all the available memory not used by the remaining node operations. This difference is also portrayed in the different DB sizes used, as depicted in Figure 3b). A second reason for the reduced performance of HBase is the fact that data retrieval is performed from the disk to the volatile memory through the Java Heap process. Thus read intensive workloads cause Heap fragmentation and are managed by the Garbage Collector, putting more strain on the system. HBase in next versions (0.96.3) gives the opportunity to the volatile memory "blockcache", which is the memory part responsible for the caching mechanism, to be decoupled from the Heap size of the RegionServer process, thus exploiting more the available memory[9]. Cassandra in its default setting does not cache data but data keys, making this their retrieval faster. Increased data volumes have caused a performance degradation of 26% in HBase, 16% in Cassandra and 19% in MongoDB (Figure 3b)



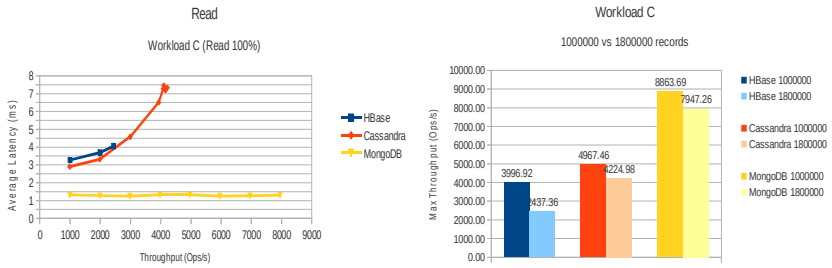
(a) (b)

Figure 4: Comparative Latency vs Throughput in YCSB Workload B

4.3 YCSB Workload C (100% Read)

In workload C (Figure 5a) , given that it is an exclusively read workload, the issues mentioned in the previous sections are clearly depicted. MongoDB achieves a very low latency due to the lack of updates and synchronization issues. On the contrary, HBase's poor caching strategy under limited available RAM has significantly affected its performance. Of course in cases where the latter does not apply, results could be improved with relation to this case. On the other hand, due to the fact that Cassandra caches the value keys mostly used, it is not affected so intensively by the data read size. However data retrieval on disk makes it less attractive than MongoDB for these

kinds of loads. For Cassandra, there is the ability to cache data rows that are requested for reads, however this option was not used since it was not part of the default settings.
As it was anticipated, the increased data volume (Figure 5b) mainly affected Hbase (39% degradation). This was significantly lower in the other DBs (17.5% for Cassandra and 11.5% for MongoDB).

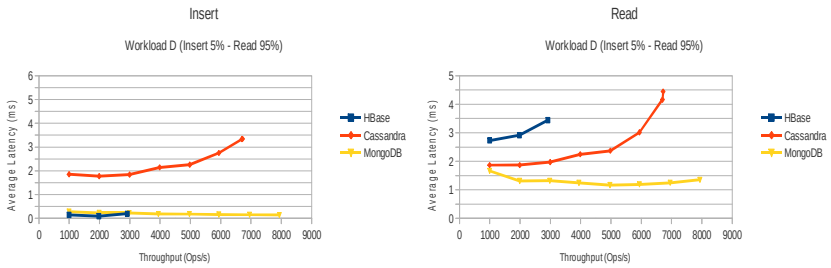


(a) (b)

Figure 5: a) Comparative Latency vs Throughput in YCSB Workload C b) Throughput Comparison for different DB sizes in Workload C

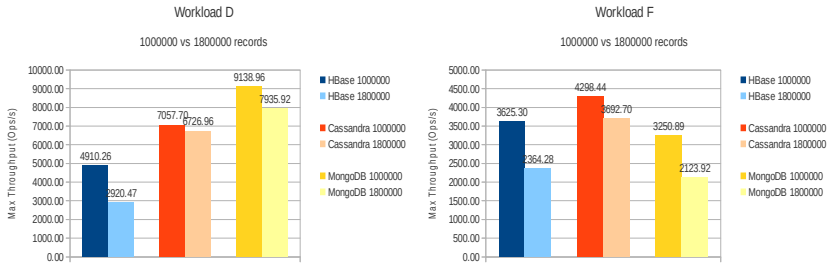
4.4 YCSB Workload D (95% Reads-5% Inserts)

In workload D, a significant difference is the fact that reads are performed with a Latest distribution on the more recently used data and not a Zipfian one. Thus MongoDB again exploits the memory mapped files and portrayes a very good performance (Figure 6). Data insertion does not seem to affect the system's performance, since MongoDB uses mechanisms to preallocate the necessary space. This on the other hand creates an issue of needing too much space for initialization of the DB. Inserts in HBase and Cassandra are performed by creating new files (HFiles and SSTables), thus eliminating the need for concrete data positioning retrieval on the disk during insertion. The usage of Latest distribution helped Cassandra to perform significantly better with relation to workload B and to approach the ratings of MongoDB. This was probably caused by the fact that an increased number of reads were served by the data contained in the system's memTable. The same behavior was expected from HBase, however the large read number has probably affected also the garbage collection in the JVM. On the contrary, in Cassandra the memory part responsible for storing the keys is off heap. HBase demonstrated a 40% drop in case of increased data volumes (Figure 7a), while for Cassandra and HBase the respective percentages were 5% and 15.1%.



(a) (b)

Figure 6: Comparative Latency vs Throughput in YCSB Workload D

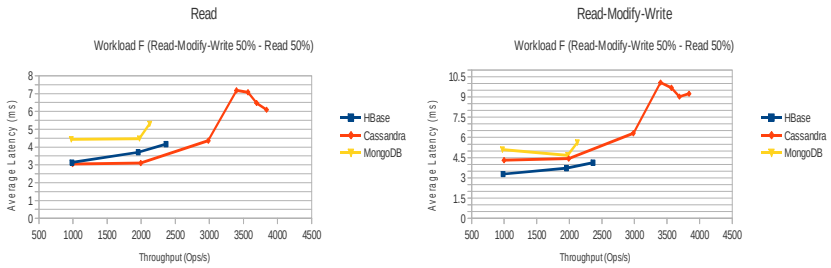


(a) (b)

Figure 7: Throughput Comparison for different DB sizes in a) Workload D and b) workload F

4.3 YCSB Workload F (50% Reads- 50% Read/Modify/Write)

In this workload, also caching mechanisms seem to be the key for reaching high performance. Due to the fact that the available RAM in each node was 4GB (much less than the recommended size), systems that invest in achieving good memory mapping are hindered by the complexity of the workload in this case. Thus MongoDB portrayed a reduced performance (similar to the one in workload A) while HBase continued to deteriorate as in the cases of B, C and D. On the other hand, Cassandra that does not try to perform these optimizations portrayed less deviation, as it appears in Figure 8. With relation to DB size (Figure 7b), HBase had a deterioration in performance by 53%, the same as MongoDB, while Cassandra was more stable portraying a 16.4% drop.

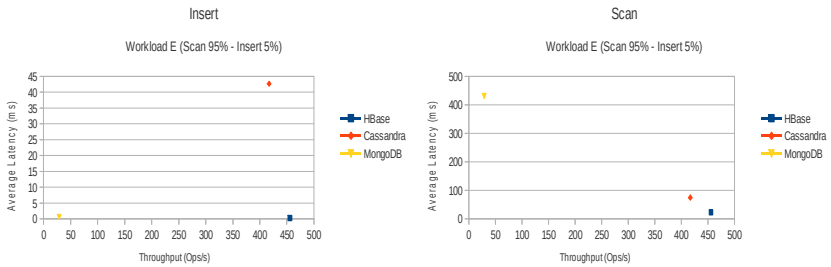


(a) (b)

Figure 8: Comparative Latency vs Throughput in YCSB Workload F

4.6 YCSB Workload E (Scan 95%- Insert 5%)

In workload E scan operation, YCSB uses a Zipfian distribution to select a specific datum and then to request the next data in a serial manner from that location. The size of this retrieval is decided via a uniform distribution with a maximum number of 100. In this case HBase and Cassandra performed better than MongoDB since their data are stored serially in files and thus their retrieval is more efficient(Figure 9) . MongoDB on the other hand performed a number of operations per file to retrieve the data from disk, if these data were not located in memory at the time of request. Another reason is that the retrievals in theDB are based on the actual data fields inside the records (due to the document orientation of Mongo) and not based on indexes like in the case of HBase and Cassandra. In terms of database size influence (Figure 10), Cassandra and HBase seem not to be affected (HBase improves actually its performance by 11%), however MongoDB suffers from an 88% drop in performance, due to the limited memory size, type of operation and data access.



(a) (b)

Figure 9: Comparative Latency vs Throughput in YCSB Workload E

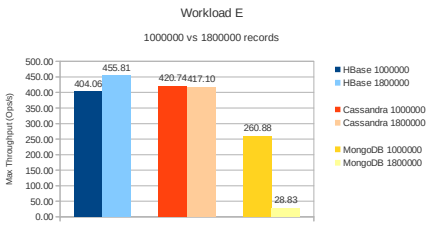


Figure 10: Throughput Comparison for different DB sizes in Workload E

5 Conclusions

In conclusion, and following the analysis performed in the previous sections, we can identify cases for which each of the investigated solutions performed optimally. MongoDB was especially efficient in cases where reads constituted the majority of performed operations thanks to the memory-mapped files that it uses. This system however exemplified a particular weakness in scan and retrieval workloads, in which disk operations were necessary. On the other hand, Cassandra's enablement of caching only for the data location on disk resulted in it being the more stable solution despite the change in workload types of the YCSB client and maintained a satisfactory and stable performance in all cases. This was enhanced by the selection of the relaxed consistency option and the usage of a triple replication setup in order to balance the workload in the system. Finally, HBase was severely affected by the choice to store temporary data on-heap especially in conjunction with the limited memory resources in the available nodes. Thus it was not only unable to exploit caching mechanisms, but the latter seemed also to hinder the efficient system operation in some cases.

The systems used in the context of this paper heavily rely on the caching mechanisms enabled in each case and the according design. The experiment aided in identifying several aspects of these mechanisms and their effect on performance, especially under limited available memory in the systems.

During setup and execution, numerous parameters of the examined tools could be toggled in order to adapt to the given tests. This kind of adaptation is worth to consider as future work, extending the insights gained from the initial examination attempted in this work and based on the envisioned usage and deployment by potential adopters of NoSQL technologies and available offerings.

Acknowledgments. This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

References

[1]. Digital Universe Infographic.IDC (Dec,2012), Available at: http://www.emc.com/infographics/digital-universe-business-infographic.htm

[2]. Presto: Interacting with petabytes of data at Facebook.Lydia Chan (Nov,2013), Available at:https://www.facebook.com/notes/facebook-engineering/presto-interacting-with-petabytes-of-data-at-facebook/10151786197628920

[3]. CERN Computing, Available at: http://home.web.cern.ch/about/computing

[4]. List of nosql databases , Available at: http://nosql-database.org

[5]. Jing Han et al, "Survey on NoSQL database," Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on , vol., no., pp.363,366, 26-28 Oct. 2011 doi: 10.1109/ICPCA.2011.6106531

[6]. Cooper, B.F., et al. "Benchmarking cloud serving systems with YCSB." In Proceedings of the 1st ACM symposium on Cloud computing, pp. 143-154. ACM, 2010.

[7]. Bonfire project Cloud testbeds: http://www.bonfire-project.eu/

[8]. Open Cloud Computing Interface Standard, available at: http://occi-wg.org/

[9]. BlockCache 101.Nick Dimiduk (Accessed: Sep-2014) [Online]:http://www.n10k.com/blog/blockcache-101/

[10]. Poess, M., and Floyd C.. "New TPC benchmarks for decision support and web commerce." ACM Sigmod Record 29.4 (2000): 64-71.

[11]. Yingjie Shi, et al. 2010. Benchmarking cloud-based data management systems. In Proceedings of the second international workshop on Cloud data management (CloudDB '10).

[12]. Hecht R, Jablonski S (2011) NoSQL evaluation: A use case oriented survey. Proc 2011 Int Conf Cloud Serv Computing:336–341

[13]. Rahman, Muntasir Raihan, et al. "Toward a principled framework for benchmarking consistency." Proceedings of the Eighth USENIX conference on Hot Topics in System Dependability. USENIX Association, 2012.

[14]. G OLAB et al , M. A. Analyzing consistency properties for fun and profit. In PODC'11: Proc. of the 30th annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (2011), pp. 197–206.

[15]. Van der Veen, Jan Sipke, et al. "Sensor data storage performance: Sql or nosql, physical or virtual." Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. IEEE, 2012.

[16]. Abadi, Daniel J. "Data Management in the Cloud: Limitations and Opportunities." IEEE Data Eng. Bull. 32.1 (2009): 3-12.

[17]. Iosup, Alexandru et al. "On the performance variability of production cloud services." Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on. IEEE, 2011.

[18]. E. A. Brewer, "Towards robust distributed systems," Symposium on Principles of Distributed Computing, 2000.

[19]. Sakr S et al. (2011) A survey of large scale data management approaches in cloud environments. IEEE Commun Surv Tutorials 13(3):311–336