Benchmarking Replication in Cassandra and MongoDB NoSQL Datastores

Gerard Haughian[1], Rasha Osman[2] and William J. Knottenbelt[1]

[1] Department of Computing, Imperial College London
London SW7 2AZ, UK
{gh413, wjk}@imperial.ac.uk
[2] Faculty of Mathematical Sciences, University of Khartoum
Khartoum, Sudan
rosman@ieee.org

Abstract. The proliferation in Web 2.0 applications has increased the volume, velocity, and variety of data sources which have exceeded the limitations and expected use cases of traditional relational DBMSs. Cloud serving NoSQL data stores address these concerns and provide replication mechanisms to ensure fault tolerance, high availability, and improved scalability. In this paper, we empirically explore the impact of replication on the performance of Cassandra and MongoDB NoSQL datastores. We evaluate the impact of replication in comparison to non-replicated clusters of equal size hosted on a private cloud environment. Our benchmarking experiments are conducted for read and write heavy workloads subject to different access distributions and tunable consistency levels. Our results demonstrate that replication must be taken into consideration in empirical and modelling studies in order to achieve an accurate evaluation of the performance of these datastores.

1 Introduction

The volume, velocity and variety of data produced and consumed by organizations in recent years has outgrown the capabilities of traditional relational DBMSs, due to the explosion of the web generated content [10]. New data stores have been designed to accommodate this emerging landscape; some of which have even been designed to work exclusively in the cloud. A main feature of these cloud data stores is horizontal scalability and high availability. Horizontal scalability is achieved through linear expansion of the data store as the workload increases. High availability is achieved through replicating the data across different machines and data centers.

NoSQL data stores use eventual consistency protocols to ensure that replicated data in some time in the future will be consistent [1]. Each data store provides consistency guarantees to (1) control how the data is distributed between the nodes of the cluster, (2) define how read and write requests are handled, (3) determine when different copies of the data are updated, and (4) specify the accepted level of consistency of the data. The *replication factor* (RF) is the

number of times a data item is duplicated across the cluster, which in most data store architectures reflects the number of physical nodes that hold a copy of the data item. The defined *consistency level* specifies how many of the replicas/nodes must respond to a request for the request to be considered valid.

Replication strategies and consistency levels impact the performance of the data store. Lower consistency levels provide lower latencies while stricter consistencies incur the overhead of inter-node communication and data passing. The performance comparison of replication and consistency guarantees is complicated by the different protocols implemented in NoSQL data stores. In this paper, we consider multi-master (Cassandra) and master-slave (MongoDB) replication and their corresponding consistency protocols.

There has been an increased interest in the benchmarking and performance of NoSQL data stores. However, the majority of the benchmarking studies in industry and academia do not consider the effect of replication in their studies. Further, different data access patterns are not investigated, as most depend on the uniform access of data and the disabling of consistency guarantees within their configurations. In contrast, this paper aims to fill a gap in the performance and benchmarking literature by presenting a benchmarking study in which we evaluate the impact of replication and consistency guarantees on the performance of Cassandra [2] and MongoDB [3]. This paper contributes the following.

- We illustrate the impact of replication on the performance of Cassandra and MongoDB NoSQL data stores using various cluster sizes in comparison to non-replicated clusters of equal sizes. Specifically, we analyze the impact of read and write heavy workloads under different levels of tunable consistency on the underlying optimizations and design decisions for each datastore.

- We provide insight into each data store's suitability to different industry applications by experimenting with three different data and access distributions, each simulating a different real-world use case.

- Our results demonstrate that replication and consistency levels have a direct impact on the performance of Cassandra and MongoDB. Therefore replication must be taken into consideration in empirical and modelling studies in order to achieve an accurate evaluation of the performance of these datastores.

This rest of this paper is organized as follows. Related work is presented in Section 2. Section 3 details the data stores benchmarked in this study. The experimental setup is described in Section 4. Benchmarking results are detailed in Section 5 and discussed in Section 6. Conclusions and future work are presented in Section 7.

2 Related Work

The development of the Yahoo! Cloud Serving Benchmark tool (YCSB) [4] has led to numerous benchmarking studies of NoSQL datastores. Cooper et al. [4]

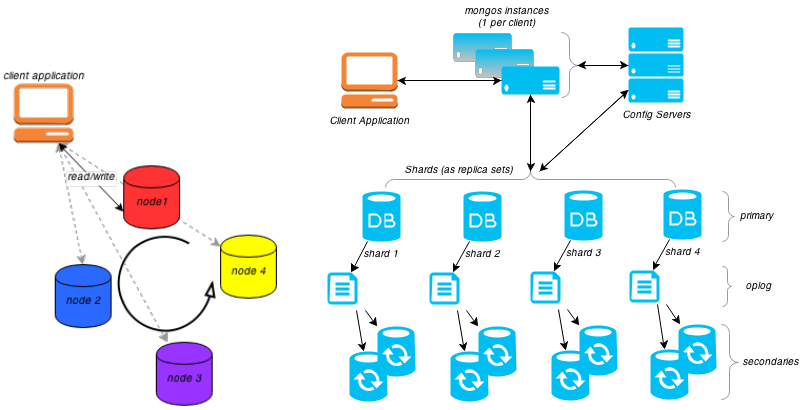
benchmarked HBase, Cassandra, PNUTS and *sharded* MySQL to illustrate the performance and scalability trade-offs of each system. Pirzadeh et al. [20] evaluated range query dominant workloads on Cassandra, HBase, and Voldemort. Rabl et al. [22] compared Redis, Cassandra and VoltDB in their ability to scale to support application performance management tools. The work in [21] compares Voldemort and Cassandra for scalability, performance and focusing on failover characteristics under different throughputs.

Dede et al. [6] evaluated the use of Cassandra for Hadoop, discussing various features of Cassandra, such as replication and data partitioning which affect Hadoop's performance. The work evaluated different replication factors with a single consistency level on clusters of up to 8 nodes. The previous benchmarking studies evaluated NoSQL datastores with non-replicated or limited replication data configurations and thus evaluating the impact of replication and different consistency levels on performance was beyond their scope. In contrast to this work, most studies assumed uniform access and data distribution which does not accurately stress the datastore.

Industrial benchmarking studies [5, 7, 16, 17, 23], configured the data stores with constant replication factors with no comparisons to baseline configurations or assessment of different access and consistency levels. Some studies tackled a very narrow problem domain (i.e., [7, 17]) by highly optimizing their studies for specific use cases or for specific data stores as in [23]. Similarly, performance modelling studies either considered configurations with no replication or replication with uniform distributions and access as in [8, 18, 19].

In this paper, we present a benchmarking study that examines the impact of replication, tuneable consistency levels and data and access distributions on the performance of two popular NoSQL datastores: Casandra and MongoDB. We investigate their performance using different replication factors selected based on the architecture of the data store using uniform, Zipfian and latest data and access distributions. We evaluate the impact of these configurations by comparing to non-replicated clusters of equal size with uniform data and access distributions.

To evaluate the effect of different consistency levels on performance we employ three different levels of consistency: (1) ONE: which indicates that only one node at most needs to reply to a request, (2) QUORUM: a specific number of nodes must reply before the request is considered valid, and (3) ALL: all nodes holding a copy of the data item must reply before a request is returned to the client. Each data store implements different replication strategies and thus these consistency levels may not be directly defined within the configuration parameters of the data store. For such cases, we have configured the data store to the closest possible configuration that produces the same level of consistency. In the following, we summarize the properties of Cassandra and MongoDB focusing on their replication strategies and consistency configurations.



(a) (b)

Fig. 1: The (a) Cassandra and (b) MongoDB architectures.

3 Systems Under Investigation

3.1 Cassandra

Cassandra is a distributed extensible record data store, developed at Facebook [11] for storing large amounts of unstructured data on commodity servers. Cassandra's architecture is a peer-to-peer distribution model [10] with no single point of failure thus supporting high availability and horizontal scalability. Data is distributed evenly across the cluster to guarantee load balancing. Cassandra offers tunable consistency settings for reads and writes, which provide the flexibility to make tradeoffs between latency and consistency [11]. For each read and write request, users choose one of the predefined consistency levels: ZERO, ONE, QUORUM, ALL or ANY. In this study, we investigated ONE, QUORUM and ALL.

Cassandra automatically replicates records throughout a cluster based on a user specified replication-factor which determines which nodes are responsible for which data ranges. Client applications can contact any node, which acts as a coordinator and forwards requests to the appropriate replica node(s) that store the data. This mechanism is illustrated in Figure 1(a). A write request is sent to all replica nodes; however the consistency level determines the number of nodes required to respond for the transaction to be considered complete. For a read request, the coordinator contacts the number of replica nodes specified by the consistency level. Cassandra is optimized for large volumes of writes as each write request is treated like an in-memory operation, while all I/O is executed as a background process. In contrast, read requests require in-memory and I/O operations in addition to consistency checks between data returned from the replicas. Keeping the consistency level low makes read operations faster as fewer replicas are checked before returning the call.

For this study, Cassandra version 1.2.16 (the latest 1.X release available before commencing this study) was used based on the supported YCSB (see Sec-

tion 4) Cassandra client driver with most of the default configurations. Hinted-handoff (a mechanism to ensure consistency of the cluster in the event of a network partition [10]) was disabled on all nodes within the cluster to avoid the hints building up rapidly within the cluster when a node fails. The tokens representing the data range for each node in each independent cluster configuration was pre-calculated and saved in separate copies of Cassandra configuration files. Finally, the RPC server type was changed to *hsha* to reduce the amount of memory used by each Cassandra node; this is ideal for scaling to large clusters. Justifications for these configurations and other Java JVM setting can be found in [9].

3.2 MongoDB

MongoDB is a document-oriented NoSQL data store that facilitates horizontal scalability by auto-partitioning data across multiple servers known as *sharding*. MongoDB's sharded architecture is represented in Figure 1(b). Each shard exists as a *replica set* providing redundancy and high availability. Replica sets consist of multiple Mongo Daemon (*mongod*) instances, including an arbiter node[3], a master node acting as the primary, and multiple slaves acting as secondaries which maintain the same data set. If the master node crashes, the arbiter node elects a new master from the set of remaining slaves.

All write operations must be directed to a single primary instance. By default, clients send all read requests to the master; however, a *read preference* is configurable at the client level on a per-connection basis, which makes it possible to send read requests to slave nodes instead [15]. Varying read preferences offer different levels of consistency guarantees. Balancing is the automatic process used to distribute the data of a sharded collection evenly across a sharded cluster which takes place within the *mongos* App server (required in sharded clusters) [14].

In this study, we used MongoDB version 2.6.1 with all standard factory settings, with the exception that journaling (i.e., logging) was disabled since the overhead of maintaining logs to aid crash recovery was considered unnecessary in this work. We setup only one configuration server which resided on the same host as a single App server. Clients interacted with this App server exclusively. It has been shown that having only one configuration server is adequate for development environments [13]. In addition, we have observed that having both servers reside on the same host did not prove to be a bottleneck.

MongoDB replication operates by way of an *oplog*, to which the master node logs all changes to its data sets. Slave nodes then replicate the master's oplog, applying those operations to their data sets. This replication process is asynchronous, therefore slave nodes may not always reflect the most up to date data. Varying *write concerns* can be issued per write operation to determine the number of nodes that should process a write operation before returning to the client

[3] An arbiter node does not replicate data and only exists to break ties when electing a new primary if necessary.

successfully. This allows for fine grained tunable consistency settings, including quorum and fully consistent writes [12].

MongoDB offers different write concerns for varying tunable consistency settings, of which NORMAL, QUORUM, and ALL write concerns where explored. MongoDB allows for concurrent reads on a collection, but enforces a single threaded locking mechanism on all write operations to ensure atomicity. In addition, all write operations need to be appended to the oplog on disk, which involves greater overhead. In contrast, regardless of the requested *read concern* no additional consistency checks are performed between replicas on read operations.

4 Experimental Setup

YCSB Configuration. The Yahoo Cloud Serving Benchmark (YCSB) [4] was developed to support benchmarking cloud NoSQL data stores. We use the YCSB benchmark to execute our benchmarking experiments on Cassandra and MongoDB. However, for the purpose of this work, we have extended its functionality as described below.

Central to the YCSB tool is the YCSB Client, which when executed in load mode inserts a user specified number of randomly generated records of size 1Kb into a specific data store with a specified distribution. In run mode, the chosen distribution determines the likelihood of certain records being read or updated. We use the following data and access distributions in the experiments, each simulating a different industry application use case [4]:

- *uniform*: items are chosen uniformly, this represents applications where the number of items associated with an event are variable, e.g., blog posts.
- *Zipfian*: items are chosen according to popularity irrespective of insertion time, this represents social media applications where popular users have many connections, regardless of the duration of their membership.
- *latest*: similar to the Zipfian distribution except items are chosen according to latest insertion time, this represents applications where recency matters, e.g., news is popular at its time of release.

In this study, one read-heavy and one write-heavy workload is used to stress the data stores. The read-heavy workload (referred to as G) is one of the default workloads provided within the YCSB Core Package; i.e., workload B comprising a 95/5% breakdown of read/write operations. The write-heavy workload (referred to as H) was custom designed to consist of a 95/5% breakdown of write/read operations. After preliminary tests, we configured the YCSB client to a fixed eight threads per CPU core, similar to [4]. For the Cassandra and MongoDB, which are not single threaded and can make use of all available CPU cores, a total of sixty-four threads were used. In order to accurately evaluate the effect of replication on data store performance, we did not increase the workload as the cluster size increased.

For MongoDB, the YCSB Client does not support write concerns or read preferences, therefore we extended the YCSB Client to facilitate this. A listing

of these extensions are given in [9]. For all experiments the primary preferred read preference was used to favor queries hitting the master, however if the master was unavailable, requests would be routed to a replicated slave. For Cassandra, the configuration for the maximum number of concurrent reads and writes was increased to match the same number of threads used by the YCSB Client, i.e., sixty-four threads.

Further, we included an additional warm-up stage to the YCSB code base to improve results and comparative analysis by using the open-source[4] warm-up extension developed for the studies in [16, 17]. Averages of the time for the data store to stabilize at or above the overall average throughput of a given experiment can be found in [9]. These warm-up times where subsequently passed as an additional configuration parameter to the YCSB Client for run phases only.

Setting	Value
OS	Ubuntu 12.04
Word Length	64-bit
RAM	6 GB
Hard Disk	20 GB
CPU Speed	2.90GHz
Cores	8
Ethernet	gigabit
Additional Kernel Settings	*atime* disabled

Table 1: Virtual Machine Specifications and Settings.

All experiments conducted in this study where carried out on a cluster of Virtual Machines (VM) hosted on a private cloud infrastructure within the same data center. Each VM had the same specifications and kernel settings as indicated in Table 1. A total of 14 VM nodes where provisioned for this study. One node was designated for the YCSB Client, and one additional node was reserved for MongoDB configuration and App servers which are required in *sharded* architectures to run on separate servers from the rest of the cluster. The remaining 12 nodes operated as standard cluster nodes which had both data stores installed but only one running at any given time. To ensure all nodes could interact effectively, each node was bound to a fixed IP address and each node was aware of the IP addresses of the other nodes.

Data Store Configuration. Each data store was configured and optimized for increased throughput, low latency, and where possible to avoid costly disk-bound operations. Each data store node hosted enough data to utilize a minimum of 80% RAM. MongoDB was configured to have a constant replication factor of two replicas per *shard*, meeting the minimum recommended production settings. The number of shards were incremented from one shard with two replicas up to 4 shards each with two replicas, in order to directly explore the write-scalability

[4]Available at `https://github.com/thumbtack-technology/ycsb`.

of MongoDB. This corresponds to cluster sizes of three nodes up to 12 nodes. Cassandra, due to its multi-master architecture, was evaluated on 3 to 12 node clusters, in which the replication factor was increased with the increase in cluster size from two to 8. For both datastores experiments were also conducted on one node clusters with no replication.

To accurately evaluate the impact of replication on datastore performance, we conducted base line experiments for comparison. These base line experiments consisted of maintaining the same cluster sizes, with no replication, using the uniform distribution only. We limited ourselves to the uniform distribution as it has been used in previous benchmarking experiments and performance modelling papers to evaluate different scenarios. Each set of experiments was repeated a minimum of three times. For each experiment: there is a warm-up phase, and the main run phase for 10 minutes and a final cool down phase. To ensure all experiments and their iterations start with the same initial state, at the end of each iteration the entire cluster is torn down and a new cluster is reconfigured and loaded with data.

5 Experimental Results

In this Section, we report the results of our benchmarking experiments. For each data store we present results of replicated clusters for each workload under different consistency levels and compare with the corresponding non-replicated baseline clusters of equal size. Confidence intervals were calculated for all results and can be found in [9], however there were too tight to appear in the graphs. In addition, due to space limitations results for read and write latencies are available in [9].

5.1 Cassandra

Throughput. From Figure 2, the effect of replication on the performance of Cassandra is very clear, as the trends of throughput for replicated clusters are directly opposite to those for non-replicated clusters. On a single non-replicated node, throughputs are 45.7% higher for the write-dominated workload (H) than the read-dominated workload (G). This is expected due to Cassandra's write optimized architecture. Further, the throughput on non-replicated clusters for workload H consistently outperforms workload G by an average of 33.1% per cluster size. In contrast, for replicated cluster sizes greater than one, we observe an average of 39.6%, 37.9%, and 30.3% decrease in throughput for the write-heavy workload (H) compared to workload G, across all cluster sizes and consistency levels for uniform, Zipfian, and latest distributions respectively. This corresponds to a 19.5%, 38.6%, and 49.7% decrease on average across all cluster sizes and distributions for ONE, QUORUM, and ALL consistency levels respectively.

Performance is most affected by the strictest consistency level ALL. This suggests that Cassandra is scalable at the cost of maintaining a lower level

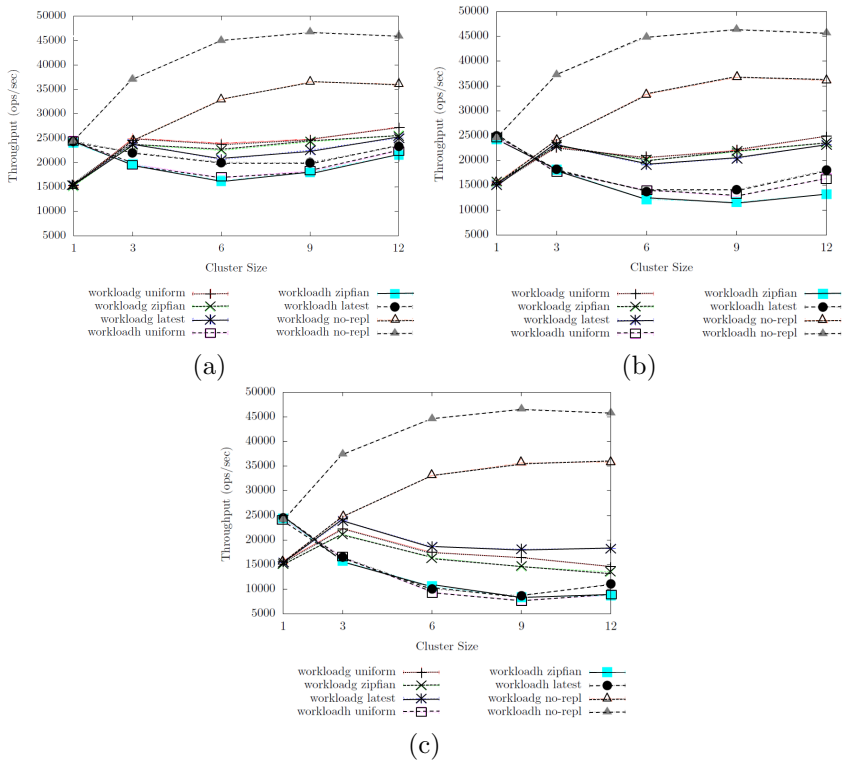


Fig. 2: Cassandra: Overall Throughputs per Consistency Level for all Workloads and Distributions: (a) ONE (b) QUORUM (c) ALL.

of consistency. However, stronger consistency levels tend to reduce scalability as the cluster size and replication factor increase. The QUORUM consistency level demonstrates a promising *sweet spot* in the consistency versus throughput tradeoff battle. Moreover, stricter consistencies have a much greater impact on write-heavy workloads than on read-heavy workloads.

Access Distributions. For workload G, we observe that the uniform distribution on average outperforms the Zipfian and latest distributions by 4.2% and 0.8% respectively. Given that the YCSB client selects a node at random for forwarding requests, this is likely to impact relative performance between distributions, favoring the uniform distribution due to a stronger correlation in their random number generators. In addition, the uniform distribution will spread the requests more evenly throughout the cluster. However, for workload H the latest distribution on average outperforms the uniform and Zipfian distributions by 7.1% and 9.7% respectively. Zipfian's poorer performance could be related to high disk access due to one key being frequently updated.

Impact of Replication. From Table 2, when comparing replicated clusters to non-replicated clusters of equal size, we observe a consistent ordering of performance metrics for both workloads based on the consistency level. For workload G, we see an average of 28.8%, 55.1% and 94.4% decrease in throughput for consistency levels ONE, QUORUM and ALL, respectively for all distributions, cluster sizes and replication factors compared to non-replicated clusters of equal size. For workload H, there is an average decrease of 74.6%, 104%, and 120.7% in throughput for consistency levels ONE, QUORUM and ALL respectively compared to non-replicated clusters of equal size. As the cluster size and replication factor increase more nodes are required to confirm each operation resulting in additional overhead and reduced performance. This trend is a reflection of Cassandra's architecture favoring availability and network partition tolerance over consistency. We note that the impact of replication on the write-heavy workload is more evident due to the overhead of updating data within the cluster.

		Workload G				Workload H			
cluster size		3	6	9	12	3	6	9	12
replication factor		2	4	6	8	2	4	6	8
Uniform	ONE	1.5	31.5	38.3	27.8	61.3	90.5	87.5	68.3
	QRM	8.6	46.4	49.6	36.8	70.5	105	114.2	129.6
	ALL	9.9	61.4	76	84.9	77.0	131.1	143.2	135.8
Zipfian	ONE	3.6	37.4	40.2	34	61.1	93.8	88.4	71.8
	QRM	7.1	49.2	51	42.2	68.8	115	120.6	110.7
	ALL	15.2	67.1	86.1	91.1	81.0	123.8	139	136.2
Latest	ONE	2.8	45.6	47.2	35.9	50.7	76.6	80	64.8
	QRM	6.3	53.2	56.7	43.6	68.3	106.6	107.3	87.1
	ALL	2.7	55.5	67.4	65.7	76.7	127	137.4	122.2

Table 2: Cassandra: The Difference (%) In Overall Throughput Between Replicated and Non-Replicated Clusters per Workload.

5.2 MongoDB

Throughput. The effect of MongoDB's contrasting consistency checks for reads and writes is evident from Figure 3 in which the throughput of the read-heavy workload (G) has on average an 89% higher level of throughput than the write-heavy workload (H). This corresponds to 94.8%, 84%, and 87.2% increases for uniform, Zipfian, and latest distributions respectively, on average across all consistency levels and cluster sizes. When broken down by consistency level, we can observe a 89.5%, 87.1%, and 89.5% increase for ONE, QUORUM, and ALL consistency levels respectively. Figure 3 illustrates how this trend varies as the cluster size increases. For both workloads we observe a performance drop from cluster sizes 1 to 3. This is due to an additional replication factor of two being

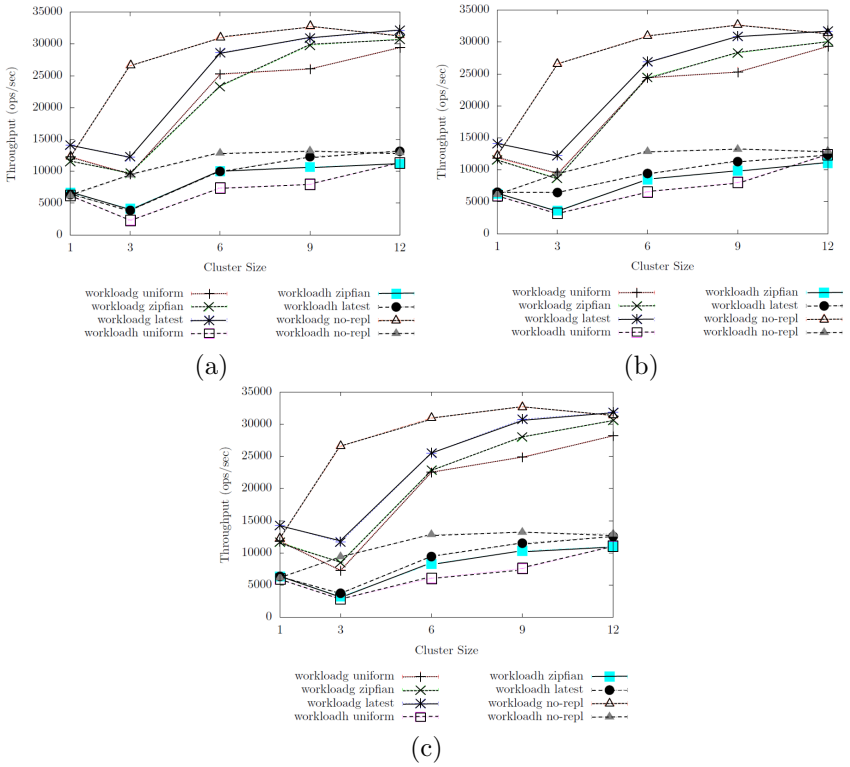


Fig. 3: MongoDB: Overall Throughputs per Consistency Level for all Workloads and Distributions: (a) ONE (b) QUORUM (c) ALL.

applied to the single shard in the 3 node cluster. The master node now needs to save data to an oplog on disk and manage two additional servers. As the cluster size increases above 3 nodes more shards distribute the load of reads and writes and thus there is an increase in throughput following the trend of the baseline non-replicated clusters of equal size.

For all subsequent cluster sizes (6+), the average decrease in throughput is only 13.6% and 40.3% for workload G and H respectively in comparison to the non-replicated clusters. This suggests that replication has a lesser effect on performance for read-heavy workloads once the overhead of maintaining a small number of shards have been overcome. When comparing based on the consistency levels, we observe higher throughputs for a consistency level of ONE on average across all distributions and cluster sizes, with slight degradations for QUORUM and ALL consistency levels.

Access Distributions. The latest distribution outperforms the Zipfian and uniform distributions for both workloads. For workload G, the latest distribution has a 15% and 17.9% increase in throughput on average across all cluster sizes and consistency levels compared to the Zipfian and uniform distributions

respectively. For workload H, the latest distribution has a 10.9% and 27.9% increase in throughput on average across all cluster sizes and consistency levels compared to Zipfian and uniform distributions respectively. This is expected as MongoDB stores all data on disk and reads data into RAM on a need to basis. The latest and Zipfian distributions would outperform the uniform distribution as accessed data would be in main memory after a short number of operations. Further, the warm-up stage added to the YCSB Client gives an added advantage to the latest and Zipfian distributions in this regard.

Impact of Replication. The impact of replication is more evident for write-heavy workloads due to the effect of consistency checks performed on writes. Table 3 shows the difference in percentages between replicated and non-replicated clusters of equal size for all experiments. From Table 3, the impact of replication on the performance of workload H in comparison to workload G, especially for cluster sizes 6+, is evident in the large differences between the throughput of workload H and that of the baseline non-replicated clusters of equal size. The effect of the access skew is clear when comparing to the baseline non-replicating clusters, as shown in Table 3. For the read-heavy workload, when comparing to the baseline non-replicated clusters of equal size, the Zipfian and latest distributions mitigate the overhead of replication due to the availability of data in main memory. This is not the case for the uniform distribution where the impact of replication is evident. When considering the write-heavy workloads, the increase in disk access on multiple replicas leads to the increased impact of replication, irrespective of access distribution, consistency level or cluster size.

		Workload G				Workload H			
cluster size		3	6	9	12	3	6	9	12
replication factor		2	2	2	2	2	2	2	2
Uniform	ONE	94.1	20.4	22.5	6.2	120.4	54.1	49.6	12.9
	QRM	95.4	23.7	25.4	6.7	99	64.8	49	15.3
	ALL	113.6	31.5	27.1	10.5	107.2	71.4	54	15.5
Zipfian	ONE	93.6	28.2	9.3	1.7	78.4	24.9	21	14.6
	QRM	101.5	23.4	14.5	4.0	89	41	29	15.7
	ALL	102.5	30.1	15.5	2.5	97.8	43	24.3	16.2
Latest	ONE	73.8	7.8	5.6	2.5	83.6	25.1	7.1	1.7
	QRM	74	14	5.7	1	37.4	30.7	15.6	4.5
	ALL	77.6	19.4	6.0	1.6	76.7	127	137.4	122.2

Table 3: MongoDB: The Difference (%) In Overall Throughput Between Replicated and Non-Replicated Clusters per Workload.

6 Discussion

Throughput. For the read-heavy workload (G), MongoDB (averaging 21230 ops/sec) is only marginally better than Cassandra (which averages 20184 ops/sec) by 5.1%. For workload H which is write dominated, the greatest difference is that Cassandra outperforms MongoDB by 72.5%. This stark contrast is a clear indication of Cassandra's write optimized architecture.

For the read-heavy workload (G), MongoDB demonstrates better performance with the latest distribution, whereas Cassandra performs best with the uniform distribution. MongoDB outperforms Cassandra on all distributions, except for the uniform distribution in which Cassandra has better throughputs than MongoDB. Cassandra's better performance on read-heavy workloads with a uniform distribution is likely a result of a strong correlation between how the YCSB Client selects a node randomly for routing requests, spreading the requests more evenly across the cluster. Whereas the latest distribution would force the same set of nodes to constantly handle operations, causing a backlog of read-repairs to build up. When accessed with the latest distribution, MongoDB is only 1.1 times more performant than Cassandra.

For the write-heavy workload (H), the latest distribution once again outperforms all other distributions on average across all cluster sizes and consistency levels, followed by Zipfian, except for Cassandra which performs second best with the uniform distribution. When all data stores are accessed with the latest distribution, Cassandra is 2 times better than MongoDB. The reason we observe larger contrasts in relative performance compared to workload G, is because Cassandra is write optimized delaying consistency checks for read time. In contrast, MongoDB performs consistency checks at write time.

Replication. To assess the impact replication on data store performance, we compare two different replication strategies, i.e., the multi-master model used by Cassandra, and the replica set model used by MongoDB. We can observe that apart from the exception of consistency level ONE on workload G, for cluster sizes 6+, MongoDB's replica set replication model has less of an impact on throughput performance than Cassandra's multi-master replication model when compared to non-replicated clusters of equal size. Cassandra's replication model accounts for a 41.1%, and 98% throughput degradation for all consistency levels and distributions, averaged across all replicated clusters sizes for workload G and H respectively. In contrast, MongoDB's replication model only accounts for 33% and 52% degradation in throughput for workload G and H respectively. This suggests that MongoDB's master-slave replication architecture has less of an effect on cluster performance than Cassandra's multi-master architecture.This is a result of each master and slave being responsible for a single data partition leading to reduced access contention compared to the multi-master model used by Cassandra in which each node contains more than one unique partition on a single server.

Performance Summary. Write-heavy workloads on non-replicated Cassandra clusters are able to exploit Cassandras write-optimized architecture. In contrast, replication has a noticeable impact on the performance of write-heavy workloads in comparison to read-heavy workloads. Cassandra is scalable at the cost of maintaining a lower level of consistency, we observed 65% and 75% degradations in performance between consistency levels ONE and ALL for read-heavy and write-heavy workload respectively. Stricter consistency levels have a greater impact (9%) on write-heavy workloads than on read-heavy workloads. Read-heavy workloads perform best when data access is random or close to random. For write-heavy workloads, memory resident datasets provide better performance (as represented by Zipfian and lastest distributions).

MongoDBs architecture is highly read-optimized, with read-heavy workloads outperforming write-heavy workloads on average by 90% across all cluster sizes, distributions and consistency levels. An interesting observation is that replication has minimal impact on performance relative to non-replicated clusters of equal size once the overhead of maintaining a small number of shards have been overcome. In addition, stricter consistency levels have on average a 5% impact on performance for both workloads. MongoDB performance is best when the entire or majority of the working data set can be kept in RAM as it would be for latest and Zipfian distributions.

7 Conclusions and Future Work

This study benchmarked replication in Cassandra and MongoDB NoSQL data stores, focusing on the effect of replication on performance compared to non-replicated clusters of equal size. To increase the applicability of this study to real-world use cases, a range of different data access distributions (uniform, Zipfian, and latest) were explored along with three tunable consistency levels: ONE, QUORUM, and ALL, and two different workloads: one read-heavy and one write-heavy. Our experiments have shown that master-slave type replication models, as exhibited by MongoDB tend to reduce the impact of replication compared to multi-master replication models exhibited by Cassandra. These results demonstrate that replication must be taken into consideration in empirical and modelling studies in order to achieve an accurate evaluation of the performance of these datastores. For future work, we plan to conduct a similar benchmarking study on the Amazon EC2 cloud, extending experiments to include larger data sets and cluster sizes while making use of solid-state disks to better reflect industry standard deployments.

References

1. P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Probabilistically bounded staleness for practical partial quorums. *Proc. VLDB Endow.*, 5(8):776–787, Apr. 2012.
2. Cassandra. `http://cassandra.apache.org/`.

3. K. Chodorow. *MongoDB: the definitive guide*. O'Reilly Media, Inc., 2013.
4. 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. ACM, 2010.
5. Datastax Coperation. Benchmarking top NoSQL databases. a performance comparison for architects and IT managers. 2013.
6. E. Dede, B. Sendir, P. Kuzlu, J. Hartog, and M. Govindaraju. An evaluation of Cassandra for Hadoop. In *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 494–501. IEEE, 2013.
7. Diomin and Grigorchuk. Benchmarking Couchbase server for interactive applications. `http://www.altoros.com/`, 2013.
8. A. Gandini, M. Gribaudo, W. J. Knottenbelt, R. Osman, and P. Piazzolla. Performance analysis of nosql databases. In *11th European Performance Engineering Workshop (EPEW 2014)*, 2014.
9. G. Haughian. Benchmarking Replication in NoSQL Data Stores. Master's thesis, Imperial College London, UK, 2014.
10. E. Hewitt. *Cassandra: the definitive guide*. O'Reilly Media Inc., 2010.
11. A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.
12. MongoDB Inc. MongoDB manual: Replication. `http://docs.mongodb.org/manual/replication/`.
13. MongoDB Inc. MongoDB manual: Sharded cluster config servers. `http://docs.mongodb.org/manual/core/sharded-cluster-config-servers/`.
14. MongoDB Inc. MongoDB manual: Sharded collection balancer. `http://docs.mongodb.org/manual/core/sharding-balancing/`.
15. MongoDB Inc. MongoDB manual: Sharding. `http://docs.mongodb.org/manual/sharding/`.
16. Nelubin and Engber. NoSQL failover characteristics: Aerospike, Cassandra, Couchbase, MongoDB. `http://www.thumbtack.net/`, 2013.
17. Nelubin and Engber. Ultra-high performance NoSQL benchmarking. `http://www.thumbtack.net/`, 2013.
18. R. Osman and P. G. Harrison. Approximating closed fork-join queueing networks using product-form stochastic petri-nets. *Journal of Systems and Software*, 110:264 – 278, 2015.
19. R. Osman and P. Piazzolla. Modelling replication in nosql datastores. In *11th International Conference on Quantitative Evaluation of Systems (QEST)*, 2014.
20. P. Pirzadeh, J. Tatemura, and H. Hacigumus. Performance evaluation of range queries in key value stores. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, pages 1092–1101. IEEE, 2011.
21. A. Pokluda and W. Sun. Benchmarking failover characteristics of large-scale data storage applications: Cassandra and Voldemort.
22. T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii. Solving big data challenges for enterprise application performance management. *Proceedings of the VLDB Endowment*, 5(12):1724–1735, 2012.
23. A. Rogers. VOLTDB in-memory database achieves best-in-class results, running in the cloud, on the YCSB benchmark. `http://tinyurl.com/VoltDB-YCSB`, May 2014. Last Accessed: June 2016.