Performance Evaluation of NoSQL Databases

Andrea Gandini[1], Marco Gribaudo[1], William J. Knottenbelt[2], Rasha Osman[2], and Pietro Piazzolla[1]

[1] Politecnico di Milano, via Ponzio 34/5, 20133 Milano, Italy
{andrea.gandini,marco.gribaudo,pietro.piazzolla}@polimi.it
[2] Department of Computing, Imperial College London, London SW7 2AZ, UK
{rosman,wjk}@imperial.ac.uk

Abstract. NoSQL databases have emerged as a backend to support Big Data applications. NoSQL databases are characterized by horizontal scalability, schema-free data models, and easy cloud deployment. To avoid overprovisioning, it is essential to be able to identify the correct number of nodes required for a specific system before deployment. This paper benchmarks and compares three of the most common NoSQL databases: Cassandra, MongoDB and HBase. We deploy them on the Amazon EC2 cloud platform using different types of virtual machines and cluster sizes to study the effect of different configurations. We then compare the behavior of these systems to high-level queueing network models. Our results show that the models are able to capture the main performance characteristics of the studied databases and form the basis for a capacity planning tool for service providers and service users.

1 Introduction

Recently NoSQL databases have become widely adopted on cloud platforms due to their horizontal scalability, schema-free data model and the capability to manage large amounts of data. Huge amounts of data require systems that are able not only to retrieve information in very short timescales, but also to scale at the same rate as the data increases. The growing importance of Big Data applications [15] has driven the development of a wide variety of NoSQL databases, e.g. Google's BigTable [5], Amazon's Dynamo [10], Facebook's Cassandra [16], Oracle's NoSQL DB [20], MongoDB [18] and Apache's HBase [14].

One of the main features of NoSQL databases is horizontal scalability [4]; that is, the capacity to scale in performance when the number of machines added to an existing cluster increases. This capability potentially wastes resources due to over-provisioning. Thus, being able to identify the correct number of nodes required for a specific workload is important. Moreover, correctly adding or removing nodes from a distributed database is often a time-consuming operation whose impact can be minimised by proper planning.

The purpose of this work is to benchmark three of the most common NoSQL databases [9], namely *Cassandra*, *MongoDB* and *HBase* in order to provide insights about their behavior under various settings. We deploy them on the Amazon EC2 cloud platform using different types of virtual machines (in term of

CPUs, memory, I/O speed, etc.) and variable number of nodes, in order to study the effect of different configurations on the performance of these systems. Finally, we present two simple high-level queuing network models that are able to capture the main features of the considered databases. These models are able to provide insight into suitable cluster sizes for NoSQL applications and form the basis of a capacity planning tool for service providers.

To date there has been limited work in benchmarking the performance of NoSQL datastores. Rabl et al. [25] benchmark six NoSQL datastores, identifying their ability to support application performance management tools. The authors report response times and throughput for workloads that scale as the number of nodes increases in a configuration. The experiments are conducted on a fixed physical hardware architecture. By contrast, the present work benchmarks three popular datastores on a visualized architecture and explores their performance under different hardware, software and workload configurations. In [8], the authors evaluate and compare the performances of four different NoSQL systems (Cassandra and HBase among them) when used for RDF processing. Their work is mainly aimed at characterizing the differences between NoSQL systems and native triple stores.

In terms of modelling efforts, the performance community has concentrated on the modelling of traditional relational databases [23], using mainly queueing networks, e.g. [19, 12, 11, 21] and more recently queueing Petri nets, e.g. [22, 7]. In [3] Mean Field analysis is used to model the replication of resources in a NoSQL application, while [1] uses a multi-formalism approach to model queries in the Apache Hive data warehousing NoSQL solution. The authors in [24], use queueing Petri nets to study the replication performance of the Cassandra NoSQL datastore. We are unaware of previous work that attempts to depict the main characteristics of multiple NoSQL datastores in one model as presented in this paper.

The remainder of this paper is organized as follows. Section 2 explains the architecture and primary characteristics of our target NoSQL databases. Section 3 discusses the experimental setup, Section 4 presents benchmarking results, and Section 5 introduces our queueing models. Finally, Section 6 concludes and considers avenues for future work.

2 NoSQL Database Architecture

Here we give a brief introduction to the main characteristics of the NoSQL databases considered, highlighting those aspects that impact on performance:

Cassandra [13] belongs to the *wide-column store* NoSQL family [26] and provides an extended key-value store method built on a column-oriented structure. Its architecture, shown in Fig. 1a, is based on a *ring* topology, in which every node is identical to the others, guaranteeing that the system has no single point of failure. Each record inserted in the database has an associated hash value called *token*. The range of tokens is partitioned among the nodes to balance the ring. Cassandra allows replication among the nodes in the cluster by duplicating

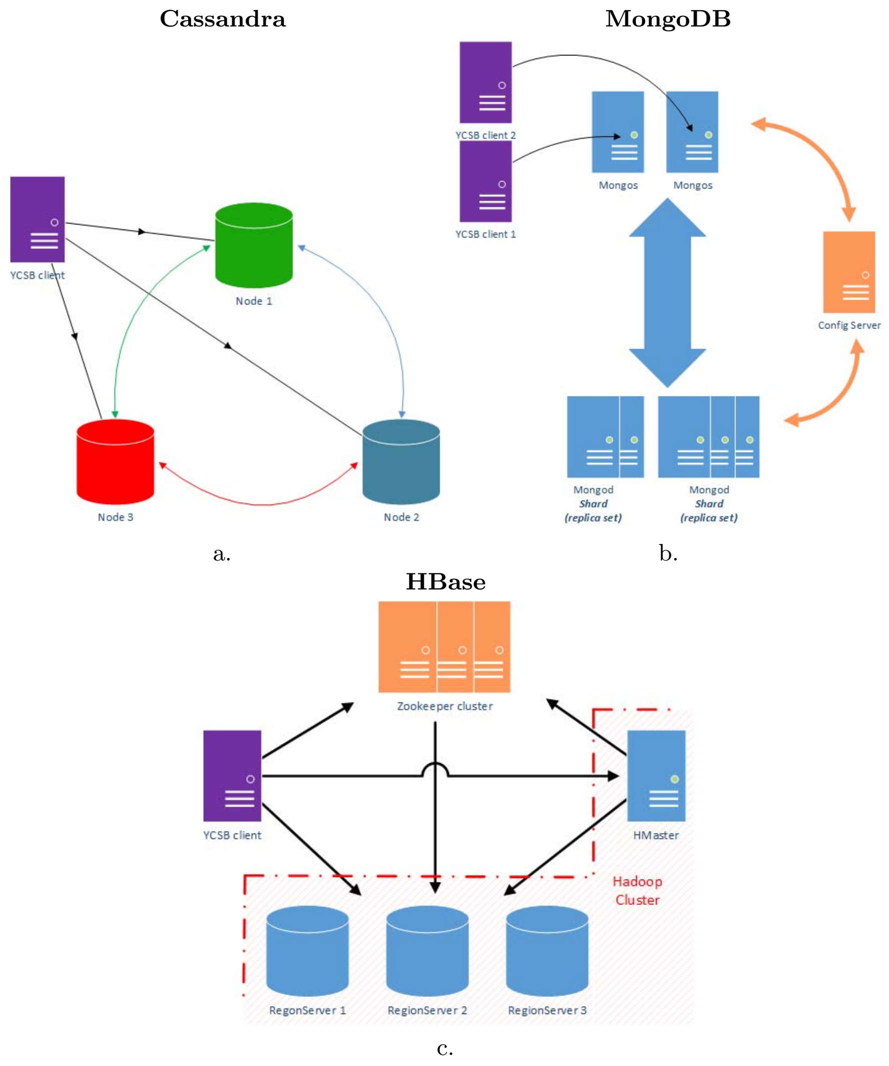


Fig. 1. The architectures of our target NoSQL Databases

data from a node to subsequent nodes on the ring. The number of replicas for each data item can be controlled by a parameter set by the user, called the *replication factor*. The *consistency level* defines the number of replicas that should respond to a data request. It is also possible to define which nodes communicate outside the ring. Such nodes are called *entry points*.

MongoDB [18] belongs to the *document-store* NoSQL family. As shown in Fig. 1b, the main components of the system are:

- `mongod`: data nodes for storing and retrieving data.
- `mongos`: the only instances able to communicate outside of the cluster.
- `config-server`: the containers of the metadata about the objects stored in the `mongod`. The metadata is used in case of a node failure. A cluster allows only one or three `config-server` instances.

Each running component constitutes one node in the MongoDB cluster.

Replication is achieved by means of *shards*. Every shard is a group of one or more nodes. The number of nodes in a shard determines the replication factor of the system: each member of the shard contains a copy of the data, creating a so-called *replica set*. Nodes belonging to the same shard have the same data. Within a shard only one node can be a master, able to execute write and read tasks; the others are considered *slave*s and can only perform read operations.

HBase [14] is based on the Hadoop map-reduce framework and Hadoop Distributed File System (HDFS). It belongs to the *wide-column store* NoSQL family. As shown in Fig. 1c, HBase relies on two supporting applications:

- `Hadoop`: a distributed map-reduce framework that provides high-throughput access to application data and which manages replication.
- `Zookeeper`: provides a distributed configuration and synchronization service for large distributed systems.

HBase has two main types of nodes: the *master*, that accounts for the nodes that are alive and which provides communication services, the zookeeper cluster and clients; and the *region servers*, which distribute data using the notion of *regions*. Regions are initially allocated to a node and split when they become too large. Thus, HBase tends to accumulate data on some nodes with a non-uniform distribution of data, especially when the system is lightly loaded.

3 Experimental Setup

We used the *Yahoo! Cloud Serving Benchmark (YCSB)* [6], a workload generator developed by `Yahoo!`, as our benchmarking framework. YCSB provides means to stress-test multiple databases and compare them in a fair and consistent way. It operates in two phases: first, data is loaded onto the data nodes (the ones responsible for storing actual data, irrespective of the naming convention of each database architecture). In our experiments, the data was generated randomly and stored by the database in its specific data format. In the second phase, YCSB executes the actual tests, in which random key requests are sent to the data nodes. The requests are randomly mixed with 50% reads and 50% writes for each client thread. The number of client threads concurrently querying the database is either varied to simulate different workload levels, or fixed to a value that saturates the DB servers. For each execution, the benchmark measures the latency (in microseconds per operation) and throughput (in operations per second).

The servers used for the tests are provided by the Amazon EC2 cloud platform. Table 1 presents the virtual hardware specifications of the machines employed for the benchmarking experiments. All nodes of a database, independently of their role in the architecture, are run on a virtual machine with the same specification during a given test to allow for a fair comparison. To ensure that the

Table 1. Virtual Hardware Specifications

Instance Type	**vCPU**	**Physical Processor**	**Memory(GiB)**
`m1.large`	2	Intel Xeon Family	7.5
`c1.xlarge`	8	Intel Xeon Family	7
`c3.2xlarge`	8	Intel Xeon E5-2680 v2	15

benchmark application does not affect the performance of the database under test, it is hosted on a dedicated machine that is not part of the DB cluster. We will refer to this machine as the *ycsb client*. We used the `m1.large` virtual machine specification for the machines hosting the nodes of the database and the `ycsb` client for most of our experiments. The `m1.large` virtual machines were chosen since they offer a good trade-off between cost and available memory. In particular, it satisfies the minimum memory requirement of HBase which is at least 7 GiB. When the number of cores per server became the focus of the tests, we switched to `c1.xlarge` instances to have a larger number of cores available. During some of the tests, due to the high number of client threads required for the total number of nodes, the `ycsb` client became the bottleneck of the system. In those cases, we switched the `ycsb` client machine to a `c3.2xlarge` instance. Every machine runs the Ubuntu Server 12.04 operating system release provided by Amazon. In order to reduce the variability inherent to the cloud environment on the measured results, some tests were performed during particular time slots when we observed that the provisioned servers showed more stability.

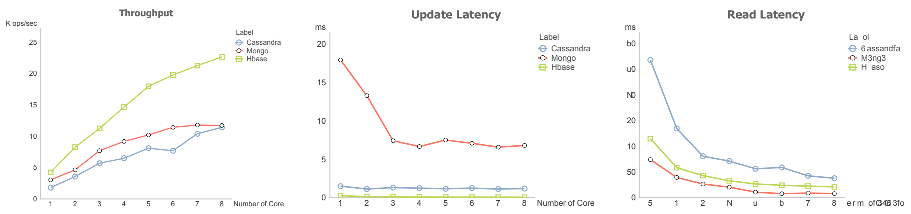
Each test is performed using a specific DB configuration in terms of active cores per server, number of nodes and data replication. The `ycsb` tool is executed at least 20 times for each configuration before collecting the average values for response time and throughput. In addition to the performance indexes collected by the benchmark, we used several `bash` scripts to track the cpu utilization of every single node in the cluster. Testing the scalability of MongoDB was not as straightforward as for the other databases. For this DB we had to experimentally determine which of its configurations was able to scale more uniformly. For MongoDB both `mongos` and `mongod` instances run on every node, while the `config-server` (that has a negligible load) runs in addition to the other two services on a randomly chosen node. However, the `ycsb` tool is not able to set more than one `mongos` connection at a time; therefore we perform the tests on MongoDB by scaling the number of clients to be equal to the number of nodes. In addition, the number of client threads running on one node is scaled by an appropriate factor to maintain the same average load used for Cassandra and HBase.

4 Benchmarking Results

Our initial tests focused on infrastructure-dependent parameters that influence the performance of a NoSQL database. We began with investigating the effect of node capacity, i.e. number of cores, on performance. The impact of the number of nodes in a cluster and the number of threads (workload) on the client was then studied. Finally, we examined the effect of the replication factor.

4.1 Number of Cores

The first set of experiments measure the database performance as the number of cores on a single node is increased. The number of client threads was fixed at 50 threads irrespective of the number of cores. These tests are performed on a single machine of the `c1.xlarge` type for both Cassandra and MongoDB, in which each was configured with one node only. For HBase, the *region server* was hosted on a single machine of the `c1.xlarge` type, and the *master* and the *zookeeper* were each hosted on a different `m1.xlarge` VM. In Fig. 2a, we show the total throughput as function of the number of cores active on a single node. It is evident that HBase is able to take advantage of the number of cores better than the other databases, scaling almost linearly. MongoDB scales well initially but then it shows a plateau at around 6 cores. Such a tendency seems absent from Cassandra which, although significantly slower that HBase in absolute terms, also scales quite linearly. Fig. 2b shows the mean update latency as function of the number of cores. In this case the mean response time required for write operations is more or less constant for both Cassandra and HBase, showing, presumably that this task is mainly I/O bound on these databases. HBase shows an extremely low update latency, benefiting from its client-side write buffer. MongoDB shows an improvement in performance when the number of cores increases from one to three, then stabilizes. This is due to the fact that the data node must communicate updates with the `config-server`. For the read operations in Fig. 2c, the behavior is more hyperbolic, as read operations are more likely to benefit from parallelization. In this case, MongoDB has the lowest latency; Cassandra shows the highest latency when running on a VM with a low number of cores. HBase has a much lower performance for read requests than for write requests since the former do not exploit a client-side buffer.

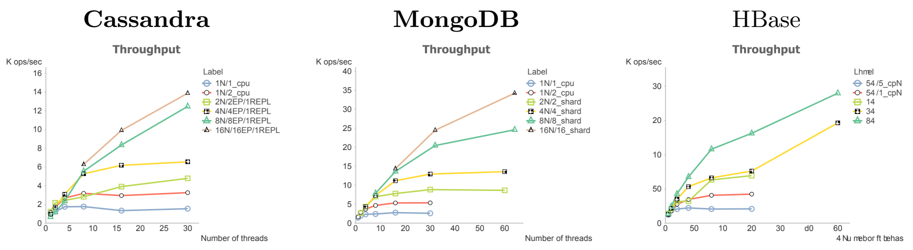


a. b. c.

Fig. 2. System throughput and latencies as a function of the number of cores

4.2 Number of Nodes

Distributed database performance is strongly affected by the size of the cluster and the workload it has to handle. To investigate the effect of the number of nodes on performance we simulate higher workload intensities by increasing the number of threads running simultaneously on the client for each request. For Cassandra and MongoDB, the number of entry points is scaled to match the number of nodes; for HBase the number of entry points is not configurable. For the single node case, we consider the performance of both on a single and a double core CPU. For more than one node, we only present results considering VMs with two cores.



a. b. c.

Fig. 3. Throughput as a function of the number of nodes

The throughput of Cassandra is considered in Fig. 3a. The throughput generally increases with an increase in the number of nodes. Fig. 3a shows the saturation point of the database, that is the point at which the database is fully utilized and throughput does not increase along with the number client threads. Configurations with 8 nodes and above do not saturate the system even at 30 threads. Similar results are shown for MongoDB in Fig. 3b and for HBase in Fig. 3c. Note that, in terms of absolute performance, HBase presents higher throughput than Cassandra and MongoDB. However, its scalability is less predictable. This is due to the non-uniform distribution of data among the nodes, in contrast to Cassandra and MongoDB.

Read and write latencies for the three DBs are reported in Fig. 4. Cassandra does not show a significant gain when the number of nodes jumps from eight to sixteen: this might be an indication of some scalability issues for very large configurations. MongoDB does not saturate with the considered workloads when distributed on four or eight nodes. HBase presents an almost linear behavior for all the considered configurations.

4.3 Replication

Replication of data in a DB cluster introduces yet another performance impacting factor. A replication factor equal to one, means that there is only one copy

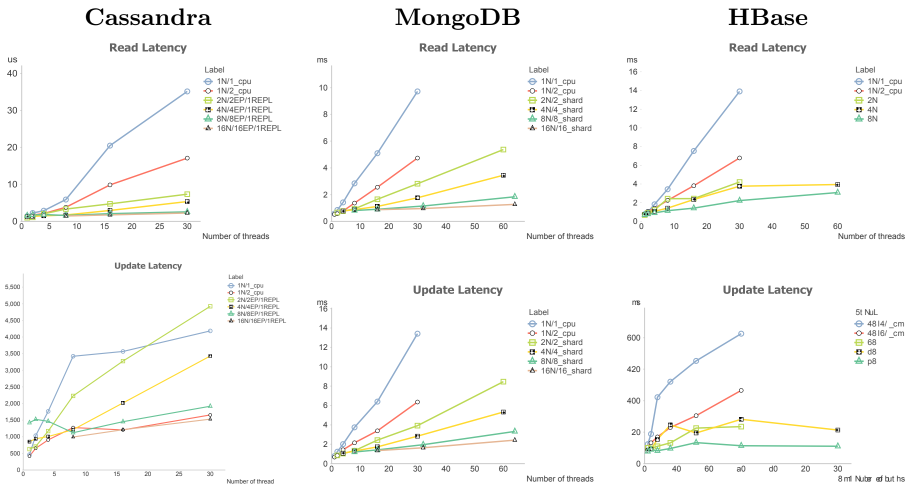
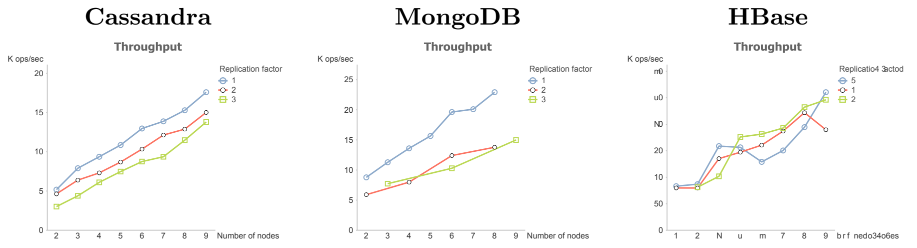


Fig. 4. Latency for the three databases with different number of nodes



a. b. c.

Fig. 5. Mean throughput for different replication factor

of the data item among all the nodes of the cluster. Higher replication factors means that, aside from the original data, copies are stored on different nodes.

Fig. 5a shows that increased replication in the cluster produces degradation of throughput. This degradation increases as more copies of the same data are requested in order to achieve consistency within the database. Fig. 5b shows the behavior of MongoDB when the replication factor is set to values higher than one. In this case, having replicas of data decreases the throughput, but differently from Cassandra, having more than one replica does not decrease performance. This is due to the fact that increase of replication in MongoDB consists in having more than one data node belonging to a shard, which basically distributes the nodes to more than one master. In Fig. 5c the effects of replication in HBase are shown. Because of the architecture of HBase, the behavior is not what was expected. There was no noted decrease in performance, nor in throughput, nor in response time. This is due to the fact that Hadoop is responsible for data allocation in a process transparent to the user. Moreover, HBase packs its data regions and splits data among its nodes randomly, hence no uniform architecture for data distribution exists.

5 Modeling a NoSQL Database

Informed by the results of Section 4, we now show how some high level features of NoSQL databases can be captured using simple queueing network models [17]. Here we present two models aimed at capturing the effect of the number of nodes in the system and of the selected replication factor.

5.1 Characterizing the Workload

We start by modelling YCSB and a system deployed on a single node (see Fig. 6). As described in Section 3, YCSB is composed of a fixed number of threads (set as a parameter of the benchmark) that concurrently perform both read and write requests with equal probability. As seen in Section 4, read and write present different performance characteristics. Moreover, write requests are subject to caching and delayed write operation, both at the client interface (HBase) and at the server (Cassandra), and thus experience a high variability in their service time.

Each client thread performs a very large number of read/write operations; hence we have modeled the entire system with a closed queueing network, where each circulating job models a YCSB thread. Read and write operations are modeled with two different classes of jobs. To reflect the random requests, we have added a class switching feature in the model of Fig. 6, such that each job has a 50% probability at each iteration of becoming of a different class. The database nodes are each modeled by two resources: one representing the CPU and the other representing the disk. Since VMs running the servers may have more than one core, the number of servers in the CPU station is set to the actual number of cores, which in this case is 2 virtual cores. The resource representing the disks are single-server with FIFO service discipline to represent the serial access that characterizes storage components. Clients are modeled by a queueing station whose number of servers is set to the number of cores of the VMs where the YCSB client are executed. This is done to reflect the resource contention of the YCSB threads. The network latency is explicitly included in the model as a infinite-server delay station as it directly affects read and write response times.

The service time of read requests is not affected by caching; therefore, read requests are executed by all the resources when submitted by the client (i.e., the network, the server CPU and the server disk). Write requests can be cached either at the client side (do not enter the network after being produced in the client, e.g., the *write buffer* in HBase), or at the server side (requests are not immediately written to the disk, as in Cassandra). The write behavior is modeled by class-dependent routing that connects the client to itself immediately after the class switching (caching at the client), and by the route that immediately returns to the client after the server cpu (caching at the server). We then determined

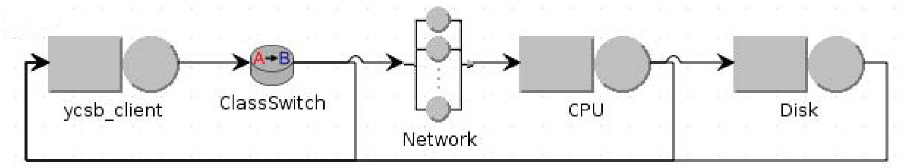


Fig. 6. Queueing network model representing a single node architecture

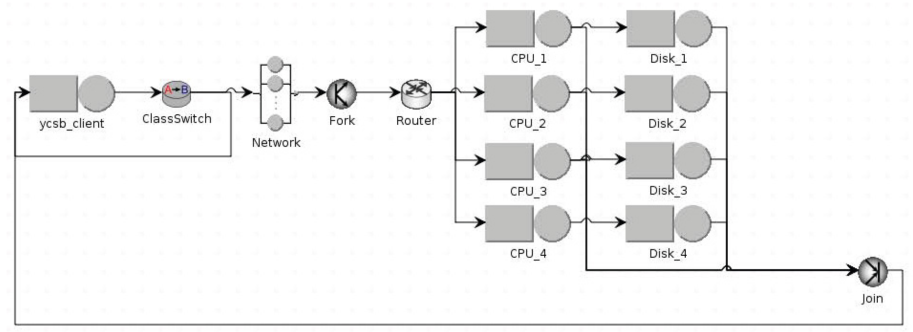


Fig. 7. Queueing network model representing a multi-node architecture

the service demands to use in the models. Network delay was measured using the *ping* command on the client machine, and estimated to be $0.45375ms$. The read and write service times for the disks were determined by benchmarking the I/O operations of the VMs and determined to be approximately $170ms$ (read) and $200ms$ (write). The CPU requirements and the caching probability are the parameters that reflect the behaviour of the considered NoSQL databases.

Tables 2, 3 and 4 show in the first column the estimated CPU demands for the read and write classes for Cassandra, MongoDB and HBase, respectively. For HBase, since it buffers write requests before sending them to the server, Table 4 has an extra column called $P(flush)$. It represents the probability that the generated request fills the buffer and that its content is sent to the server by routing it through the route that connects the class-switch node to the network delay. With probability $1 - P(flush)$, requests are not sent to the servers, so the corresponding job is immediately routed back to the client. For the other DBs, since they do not have client-side buffering, $P(flush) = 1$. To include server side caching of write commands, the probability of accessing the disk has been set to 10% on the DB: i.e. 90% of the write requests are served by the CPU only, and immediately return to the client.

The models have been analyzed using the JMT (Java Modeling Tool) [2]. The mean total throughput and mean read and write throughput for the single node configuration are shown in Fig. 8 for Cassandra, MongoDB and HBase. The `JSimGraph` component was used to solve the models by discrete event simulation. Confidence intervals were set to 95%; the resulting confidence intervals were too

Table 2. Model Parameters for Cassandra

node(s)	replication factor	cpu demand (write)	cpu demand (read)
1	-	0.075	1.15
4	1	0.8	1.25
4	4	0.15	0.55

Table 3. Model Parameters for MongoDB

node(s)	replication factor	cpu demand (write)	cpu demand (read)
1	-	0.42	0.28
4	1	0.7	0.3
4	4	0.42	0.9

Table 4. Model Parameters for HBase

node(s)	cpu demand (write)	cpu demand (read)	P(flush)
1	0.25	0.43	0.03
4	0.2	0.87	0.01

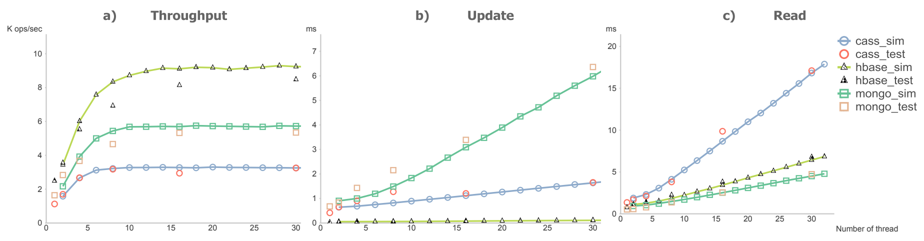


Fig. 8. Mean throughput and latency for one node for Cassandra, MongoDB and HBase

tight to appear on the graphs. Table 5 shows the relative error in the first line. From the results, the model captures the system mean throughput and the mean response times of the read and of the write operations for both Cassandra and MongoDB. For HBase, the model has a large error for the throughput. HBase is in fact the most complex of the three DBs, so a simple queuing network cannot accurately capture all its features and produce acceptable results.

5.2 Configurations with Multiple Nodes

We then model the configurations with more than one node with the queueing network model, as shown in Fig. 7. In this case we have several groups of two

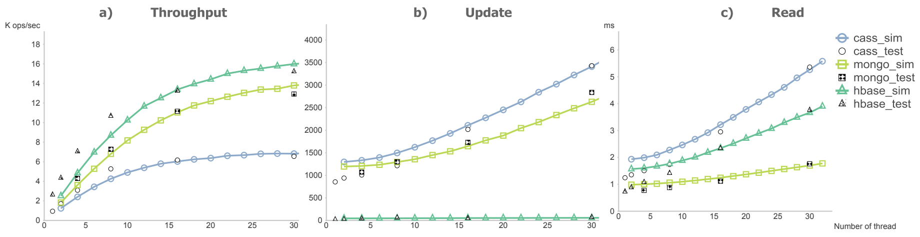


Fig. 9. Mean throughput and latency for four nodes and replication factor set to one for Cassandra, MongoDB and HBase

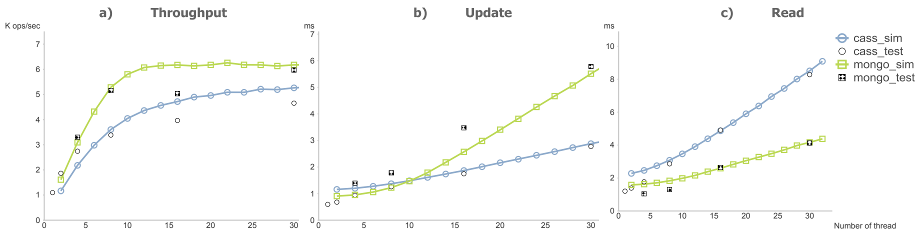


Fig. 10. Mean throughput and latency for four nodes and replication factor set to four for Cassandra, MongoDB and HBase

Table 5. Relative errors of the proposed models: throughput (**th**), write latency (**wr**), read latency (**re**)

node(s) repl.	**Cassandra (th/wr/re)**	**MongoDB (th/wr/re)**	**HBase (th/wr/re)**
1 / 1	4% / 14% / 8.4%	12.2% / 16.9% / 21.9%	24.4% / 10.7% / 9%
4 / 1	15.1% / 19.7% / 23.4%	7.5% / 6.3% / 16.3%	19.6% / 33% / 30.5
4 / 4	19.3% / 23.8% / 22.4%	8.7% / 23.4% / 2.4%	N.A.

queueing stations per node, representing respectively the CPU and the disk. A router models the choice of the node that will serve the requests: for Cassandra and HBase it implements a random selection policy. For MongoDB, it behaves differently for the read and for the write class. Read requests are randomly sent to the nodes. Write requests are sent to a single node (the topmost node of the model) to represent single-master replication of this specific database. We assume that network latency is the same for all the nodes: in this way we can model the network delay with a single delay station inserted before the router. Since the presence of more nodes introduces extra overhead in the computations necessary to serve the requests, we have to estimate a different service demand for each of the considered DB, as shown in the second line of Tables 2, 3 and 4. Fig. 9 compares the model results with the measurement of Section 4, and the second line of Table 5 reports the relative errors. The model performs quite well for high loads, but it presents larger errors for a small number of client threads.

Finally we consider the effect of replication in both Cassandra and MongoDB: we do not include HBase since that DB does not have a specific replication feature, and instead relies on the underlying Hadoop infrastructure. We model replication with the *fork and join* feature of queueing networks (nodes *Fork* and *Join* in Figure 7). We fork each job based on the number of replicas, and join them again before continuing to the client. In this case, we have to estimate the demands for the CPU operations to account for the overheads required to consider this additional feature. These corresponding values are presented in the last line of Tables 2 and 3. Results are compared with measurements in Fig. 10 and relative errors are reported in the last line of Table 5. The results are acceptable for high loads, but are inaccurate for the lower loads. In this case, replication at high loads produces high replica requests between nodes causing bottlenecks similar to the previous case. The model inaccuracies at low loads is due to the fact that the fork and join feature approximates replication synchronization, as it is not symmetric as in the fork and join queue. Therefore, the model sends requests to all nodes, which in reality does not happen.

6 Conclusion

This paper has presented benchmarks and models for three of the most common NoSQL databases: Cassandra, MongoDB and HBase. We deployed them on the Amazon EC2 cloud platform using different types of virtual machines and cluster sizes to study the effect of different configurations and to characterize the performance behavior of the databases. Using a high-level queueing network model we represented these characteristics. Our results showed that the models are able to capture much of the main performance characteristics of the studied databases at high workloads. Further investigation into modelling complex replication is required to accurately reflect the performance of replicated data. Future work includes benchmarking other NoSQL databases and providing a generic modelling framework.

References

[1] Barbierato, E., Gribaudo, M., Iacono, M.: Performance evaluation of nosql big-data applications using multi-formalism models. Future Generation Computer Systems (2013) (to appear) (available online)

[2] Bertoli, M., Casale, G., Serazzi, G.: JMT: Performance engineering tools for system modeling. SIGMETRICS Perform. Eval. Rev. 36(4), 10–15 (2009)

[3] Castiglione, A., Gribaudo, M., Iacono, M., Palmieri, F.: Exploiting mean field analysis to model performances of big data architectures. Future Generation Computer Systems (2013) (article in press); cited by (since 1996)

[4] Cattell, R.: Scalable sql and nosql data stores. SIGMOD Rec. 39(4), 12–27 (2011)

[5] Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst. 26(2), 4:1–4:26 (2008)

[6] Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with ycsb. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, pp. 143–154. ACM, New York (2010)

[7] Coulden, D., Osman, R., Knottenbelt, W.J.: Performance modelling of database contention using queueing petri nets. In: ICPE, pp. 331–334 (2013)

[8] Cudré-Mauroux, P., et al.: NoSQL databases for RDF: An empirical evaluation. In: Alani, H., et al. (eds.) ISWC 2013, Part II. LNCS, vol. 8219, pp. 310–325. Springer, Heidelberg (2013)

[9] Db engines. Db-engines ranking of database management systems (March 2014) (accessed: March 04, 2014)

[10] De Candia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. SIGOPS Oper. Syst. Rev. 41(6), 205–220 (2007)

[11] Di Sanzo, P., Palmieri, R., Ciciani, B., Quaglia, F., Romano, P.: Analytical modeling of lock-based concurrency control with arbitrary transaction data access patterns. In: WOSP/SIPEW 2010, pp. 69–78. ACM, New York (2010)

[12] Elnikety, S., Dropsho, S., Cecchet, E., Zwaenepoel, W.: Predicting replicated database scalability from standalone database profiling. In: EuroSys 2009, pp. 303–316. ACM, New York (2009)

[13] Apache Software Foundation. Cassandra, `http://cassandra.apache.org/` (accessed: March 04, 2014)

[14] Apache Software Foundation. Hbase project, `https://hbase.apache.org/` (accessed: March 04, 2014)

[15] Labrinidis, A., Jagadish, H.V.: Challenges and opportunities with big data. Proc. VLDB Endow. 5(12), 2032–2033 (2012)

[16] Lakshman, A., Malik, P.: Cassandra: A decentralized structured storage system. SIGOPS Oper. Syst. Rev. 44(2), 35–40 (2010)

[17] Lazowska, E.D., Zahorjan, J., Graham, G.S., Sevcik, K.C.: Quantitative System Performance. Prentice-Hall (1984)

[18] MongoDB, Inc. Mongodb, `http://www.mongodb.org/` (accessed: March 04, 2014)

[19] Nicola, M., Jarke, M.: Performance modeling of distributed and replicated databases. IEEE Trans. on Knowl. and Data Eng. 12(4), 645–672 (2000)

[20] Oracle. Oracle nosql database. An oracle white paper. white paper (September 2011)

[21] Osman, R., Awan, I., Woodward, M.E.: Queped: Revisiting queueing networks for the performance evaluation of database designs. Simulation Modelling Practice and Theory 19(1), 251–270 (2011)

[22] Osman, R., Coulden, D., Knottenbelt, W.J.: Performance modelling of concurrency control schemes for relational databases. In: Dudin, A., De Turck, K. (eds.) ASMTA 2013. LNCS, vol. 7984, pp. 337–351. Springer, Heidelberg (2013)

[23] Osman, R., Knottenbelt, W.J.: Database system performance evaluation models: A survey. Perform. Eval. 69(10), 471–493 (2012)

[24] Osman, R., Piazzolla, P.: Modelling replication in nosql datastores. In: QEST (2014)

[25] Rabl, T., Sadoghi, M., Jacobsen, H.-A., Gómez-Villamor, S., Muntés-Mulero, V., Mankowskii, S.: Solving big data challenges for enterprise application performance management. PVLDB 5(12), 1724–1735 (2012)

[26] Weber, S.: Nosql databases. University of Applied Sciences HTW Chur, Switzerland