Benchmarking Top NoSQL Databases

A Performance Comparison for Architects and IT Managers

White Paper

BY DATASTAX CORPORATION

FEBRUARY 2013

Contents

DATASTAX

Introduction

NoSQL databases claim to deliver faster performance than legacy RDBMS systems in various use cases, most notably those involving big data. While this is oftentimes the case, it should be understood that not all NoSQL databases are created alike where performance is concerned. This being the case, system architects and IT managers are wise to compare NoSQL databases in their own environments using data and user interactions that are representative of their expected production workloads before deciding which NoSQL database to use for a new application.

Another way to understand performance tradeoffs between different NoSQL databases is to review independent benchmarks that are produced that compare each database under different workloads. While such tests can never take the place of proof of concepts done using the exact use cases and infrastructure that a new application is targeting, they can be useful to understand the general strengths and weaknesses of a database under various workloads.

After engineers at the University of Toronto conducted a 2012 benchmark finding Apache Cassandra the "clear winner throughout our experiments" , DataStax asked End Point Corporation, a database and open source consulting company, to perform a benchmark of three top NoSQL databases – Apache Cassandra, Apache HBase, and MongoDB – using a variety of different workloads on AWS. This paper documents the results obtained by End Point and provides details on how each database performed.

Abstract

Fast performance is key for nearly every data-driven system, and IT professionals work hard to ensure that the database they select is optimized for the success of their application use cases. The best way to evaluate platforms is to conduct a formal Proof-of-Concept (POC) in the environment in which the database will run, under anticipated data and concurrent user workloads.

POC processes that include the right benchmarks, such as configurations, parameters and workloads, give developers and architects powerful insight about the platforms under consideration. This paper examines the performance of the top 3 NoSQL databases using the Yahoo Cloud Serving Benchmark (YCSB).

Benchmark Configuration and Results Summary

End Point conducted the benchmark on Amazon Web Services EC2 instances, which is an industry-standard platform for hosting horizontally scalable services such as the three NoSQL databases that were tested. In order to minimize the effect of AWS CPU and I/O variability, End Point performed each test 3 times on 3 different days. New EC2 instances were used for each test run to further reduce the impact of any "lame instance" or "noisy neighbor" effect on any one test.

Benchmark Configuration

The tests ran in the cloud on Amazon Web Services (AWS) EC2 instances, using spot instances to ensure cost efficiency while getting the same level of performance. The tests ran exclusively on m1.xlarge size instances (15 GB RAM and 4 CPU cores) using local instance storage for performance. The m1.xlarge instance type allows for up to 4 local instance devices; the instances were allocated all 4 block devices, which were then combined on boot into a single 1.7TB RAID-1 volume.

The instances use customized Ubuntu 12.04 LTS AMI's with Oracle Java 1.6 installed as a base. On start up, each instance calls back to a parent instance for its configuration. A customized script was written to drive the benchmark process, including managing the start up, configuration, and termination of EC2 instances, calculation of workload parameters, and driving the clients to run the tests.

The YCSB test was utilized for the various benchmark workloads.The client AMI's had the latest git checkout of YCSB built and installed. The AMI used was built with the `ZIPFIAN_CONSTANT` in core/src/main/java/com/yahoo/ycsb/generator/ZipfianGenerator.java set to the default 0.99.

Cassandra Configuration

The DataStax AMI utilized Apache Cassandra 1.1.6. It received its node ID and initial token value from the head node on start up. The initial token is calculated as (node ID) * 2127/(node count) to evenly distribute the token ranges across the shards. Additionally the first node was declared the contact point, and set as the seed_provider so that the hosts can find each other.

1 *Solving Big Data Challenges for Enterprise Application Performance Management*, Tilman Rable, et al., August 2012

HBase Configuration

The HBase AMI had the latest stable Hadoop (1.1.1) installed via package from http://archive.apache.org/dist/hadoop/core/hadoop-1.1.1/, though as no HBase package could be found the latest stable (0.94.3) was installed by source. It received its node ID from the head node on start up. The first node declared itself the master and runs as the Hadoop namenode and the Zookeeper. All nodes then start as a datanode and regionserver, pointing at the master node for coordination.

Once the cluster was ready, the driver script connected to the first node and created its objects. If the cluster only contained a single data node the script ran: "`create 'usertable', 'data'`", but otherwise the script pre-split the regions by running:

```
create 'usertable', 'data', {{NUMREGIONS => (node count), SPLITALGO =>
'HexStringSplit'}}
```

MongoDB Configuration

The MongoDB AMI has the latest stable release (2.2.2) installed via packages from http://downloads-distro.mongodb.org/repo/ubuntu-upstart. The package only includes an init script for the base mongod process, so additional scripts were written to start the mongos shard process and a second mongod to serve as the configuration server if needed. It received its node ID from the head node on start up. The first node declared itself the configuration server and started the mongod instance. All nodes then started mongodb, pointing at the configuration server and ran "`sh.addShard()`" on itself.

Once the cluster was ready, the driver script connected to the first node and marked the collection to enable sharding as:

```
sh.enableSharding("ycsb")
sh.shardCollection("ycsb.usertable", { "_id": 1})
```

Then, to pre-distribute the chunks by ID the script ran the following for each shard:

```
db.runCommand({{split:"ycsb.usertable", middle: {{_id: "user(calculated
range)"}}}})
db.adminCommand({{moveChunk: "ycsb.usertable", find:{{_id: "user(calcu-
lated range)"}}, to: "shard(node ID)"}})
```

Tested Workloads

The following workloads were included in the benchmark:

1. Read-mostly workload, based on YCSB's provided workload B: 95% read to 5% update ratio
2. Read/write combination, based on YCSB's workload A: 50% read to 50% update ratio
3. Write-mostly workload: 99% update to 1% read
4. Read/scan combination: 47% read, 47% scan, 6% update
5. Read/write combination with scans: 25% read, 25% scan, 25% update, 25% insert
6. Read latest workload, based on YCSB workload D: 95% read to 5% insert
7. Read-modify-write, based on YCSB workload F: 50% read to 50% read-modify-write

2 Edgar F. Codd, Wikipedia.org, http://en.wikipedia.org/wiki/Edgar_F._Codd.

Testing Parameters

In all cases `fieldcount` was set to 20 to produce 2 KB records. In all workloads except workload6 (read-latest) the request distribution was set to "zipfian"; in workload6 request distribution was set to "latest". The workloads that involve scans (4 and 5) max scan length were limited to 100, and max execution time was limited to 1 hour. All other workloads were run without a time limit, instead using an operation count of 900,000.

Other parameters were calculated at test run time by the driver script. The record count was calculated based on the number of requested data instances for the test to generate 15M records (30GB) per data node, to specifically target a dataset twice the size of available RAM.

The benchmark had a client instance for every 3 data instances (rounded up.) Insert count for each set of client instances was then calculated by distributing the record count across the client instances. The tests also targeted 128 client threads per data instance (32 per CPU core) and distributed these across the client instances.

The calculations above work out to these values:

Date Nodes	Client Nodes	Total Records	Records/Client	Total Threads	Threads/Client
1	1	15M	15M	128	128
2	1	30M	30M	256	256
4	2	60M	30M	512	256
8	3	120M	40M	1024	341
16	6	240M	40M	2048	341
32	11	480M	43.6M	4096	372

Results Summary

From an operations-per-second/throughput perspective, Cassandra proved to outdistance the performance of both HBase and MongoDB across all node configurations (1 to 32 nodes).

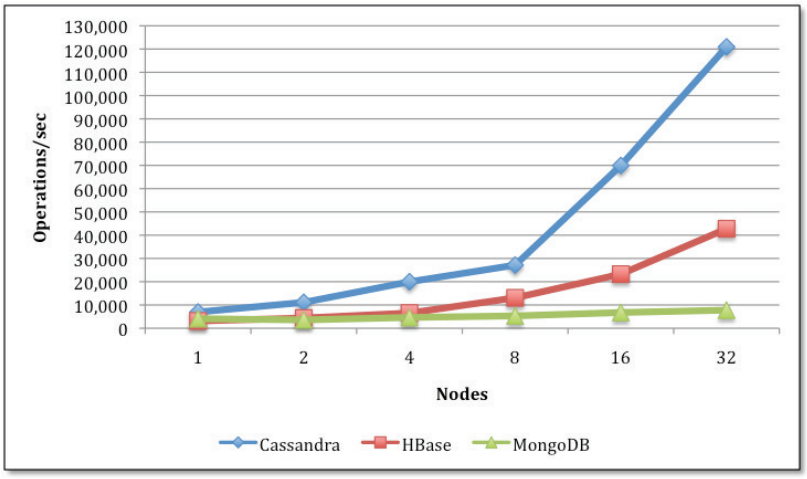
Latency metrics for Cassandra proved to be the lowest across all workloads in the tests.

Benchmark Configuration and Results Summary

Throughput Results

For throughput /operations-per-second (**more is better**), the raw numbers along with the percentage at which Cassandra outpaced HBase and MongoDB follows each graph.

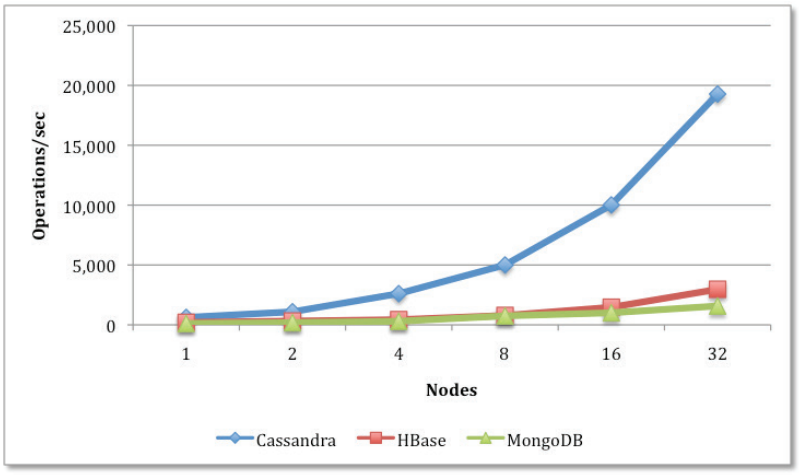
Load process



Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	6955.33	3126.29	4101.73	122.48%	69.57%
2	11112.07	4379.17	3572.88	153.75%	211.01%
4	19965.86	6528.82	4630.18	205.81%	331.21%
8	27131.40	13067.41	5307.89	107.63%	411.15%
16	69922.85	23250.44	6767.80	200.74%	933.17%
32	120918.17	42761.57	7777.36	182.77%	1454.75%

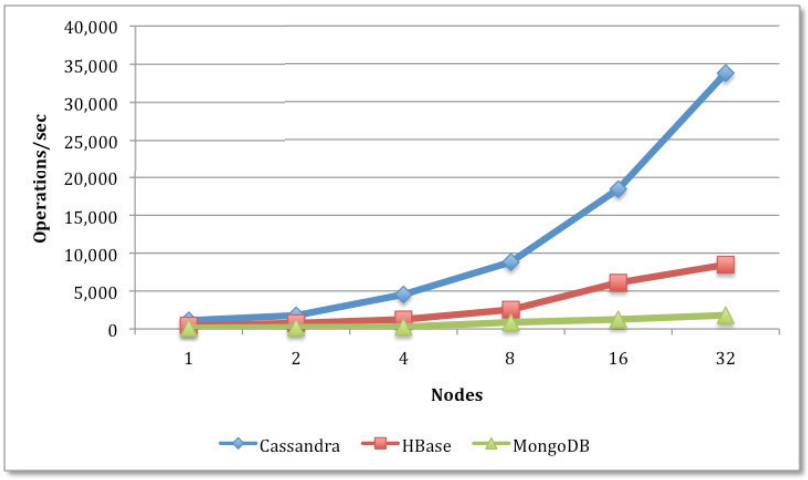
For the load process, MongoDB was not able to scale effectively to 32 nodes and produced errors until the thread count was reduced to 20.

Read-mostly Workload



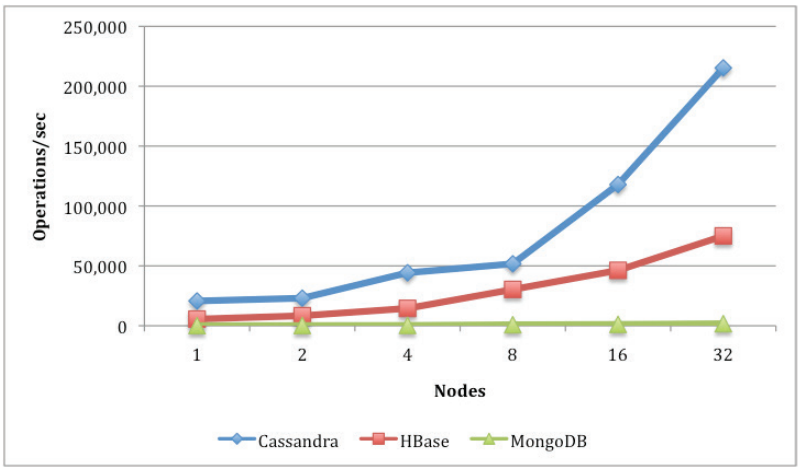
Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	624.31	206.06	115.47	202.97%	440.67%
2	1091.83	318.18	183.03	243.15%	496.53%
4	2608.90	433.89	298.69	501.28%	773.45%
8	5000.98	784.60	746.26	537.39%	570.14%
16	10016.62	1467.96	1006.41	582.35%	895.28%
32	19277.25	2969.33	1575.21	549.21%	1123.79%

Read/Write Mix Workload



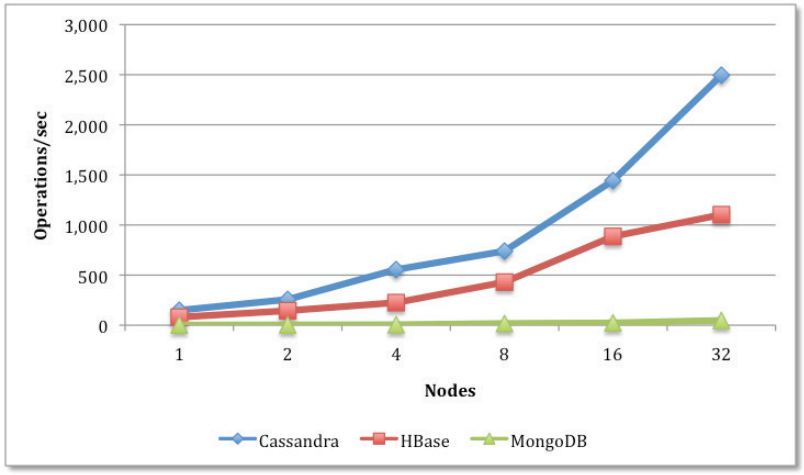
Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	1118.90	448.86	122.80	149.28%	811.16%
2	1781.36	762.83	195.85	133.52%	809.55%
4	4543.78	1249.72	268.54	263.58%	1592.03%
8	8827.11	2542.40	869.21	247.20%	915.53%
16	18454.53	6101.27	1257.67	202.47%	1367.36%
32	33767.99	8477.16	1807.26	298.34%	1768.46%

Write-mostly Workload



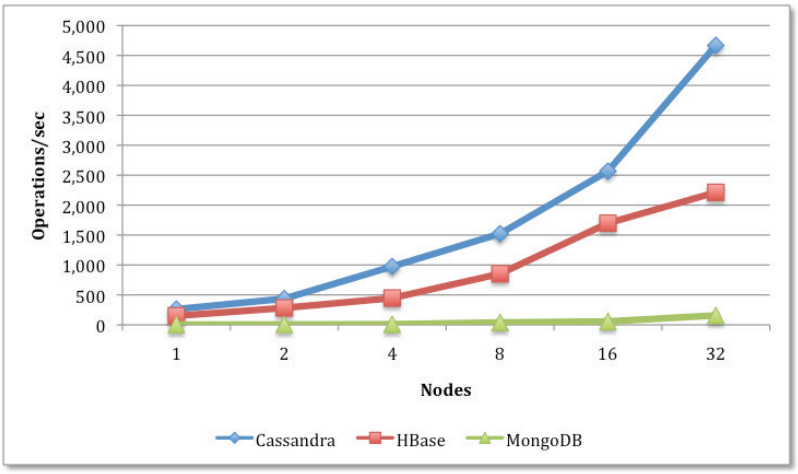
Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	20692.26	5548.53	123.18	272.93%	16698.39%
2	23032.20	8358.53	185.97	175.55%	12284.90%
4	44253.28	14493.68	277.38	205.33%	15854.03%
8	51718.85	30328.45	865.66	70.53%	5874.50%
16	117909.48	46251.65	1148.93	154.93%	10162.55%
32	215271.59	75008.83	1715.92	186.99%	12445.55%

Read/Scan Mix Workload



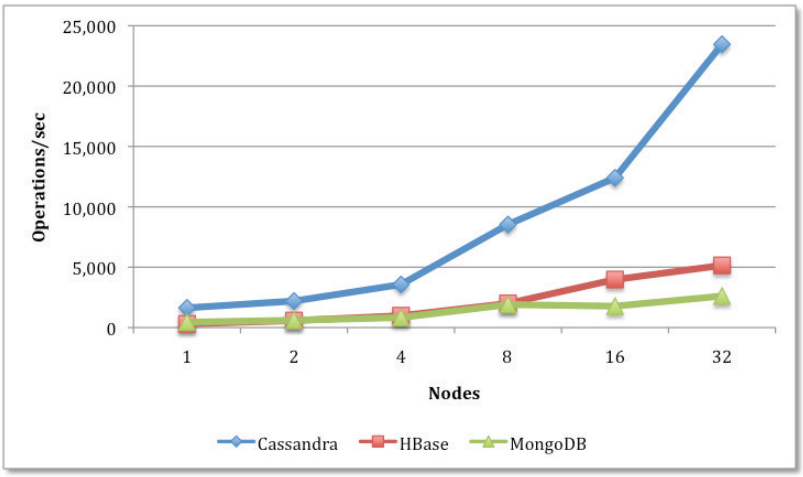
Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	149.09	80.19	1.61	85.92%	9160.25%
2	256.95	144.90	3.33	77.33%	7616.22%
4	555.20	225.41	5.06	146.31%	10872.33%
8	739.59	430.03	17.39	71.99%	4152.96%
16	1443.29	886.22	23.35	62.86%	6081.11%
32	2495.07	1103.07	47.71	126.19%	5129.66%

Write/Scan Mix Workload



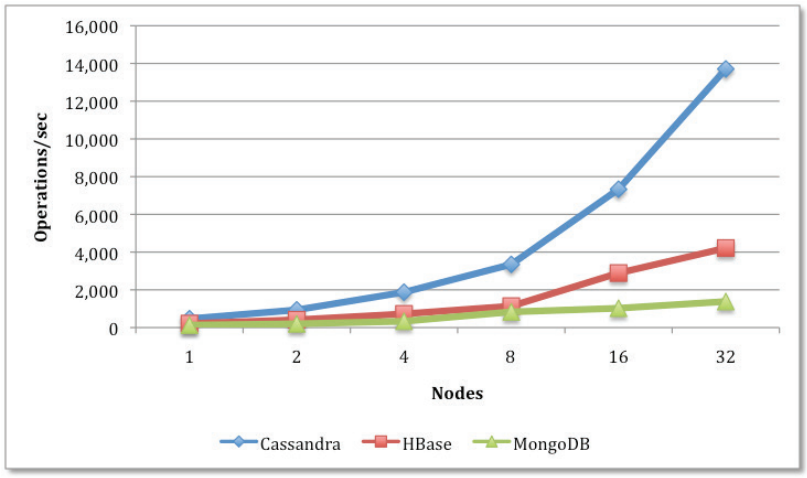
Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	261.75	150.27	3.49	74.19%	7400.00%
2	437.15	282.82	6.67	54.57%	6453.97%
4	974.57	447.27	10.19	117.89%	9463.98%
8	1522.25	853.03	39.74	78.45%	3730.52%
16	2568.09	1698.38	57.20	51.21%	4389.67%
32	4668.04	2213.02	157.30	110.94%	2867.60%

Read-latest Workload



Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	1632.87	300.57	445.68	443.26%	266.38%
2	2196.98	580.01	604.21	278.78%	263.61%
4	3567.77	981.69	829.58	263.43%	330.07%
8	8534.39	1998.04	1902.02	327.14%	348.70%
16	12402.94	3976.66	1772.83	211.89%	599.61%
32	23462.98	5151.08	2611.50	355.50%	798.45%

Read-Modify-Write Workload

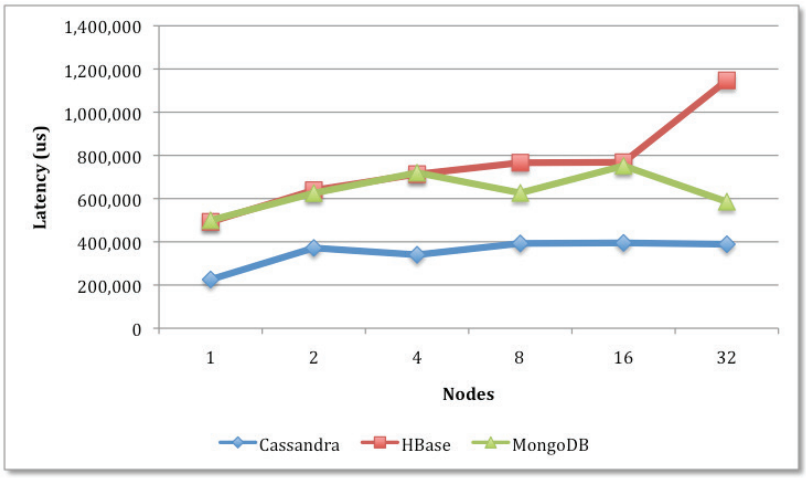


Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	467.34	210.73	126.78	121.77%	268.62%
2	937.64	412.90	196.89	127.09%	376.23%
4	1873.46	725.07	335.35	158.38%	458.66%
8	3347.56	1136.38	831.00	194.58%	302.84%
16	7333.63	2889.74	1025.19	153.78%	615.34%
32	13708.59	4213.97	1381.47	225.31%	892.32%

Latency Results

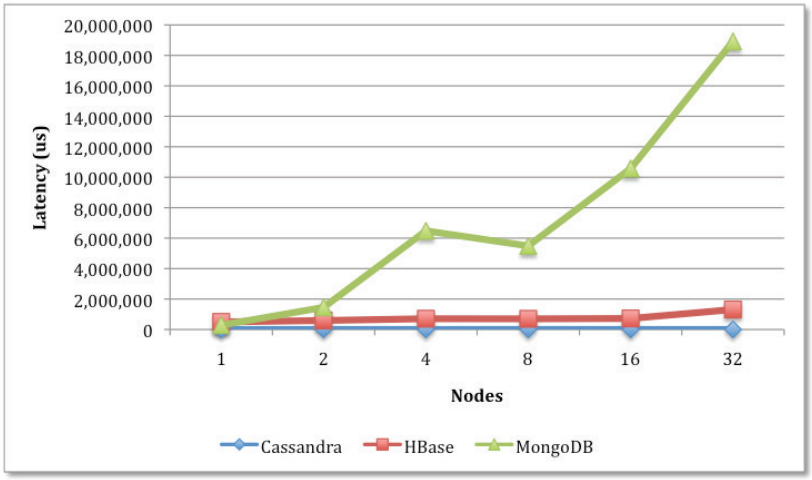
The following latency results **(less is better)** were witnessed for each test:

Read latency across all workloads



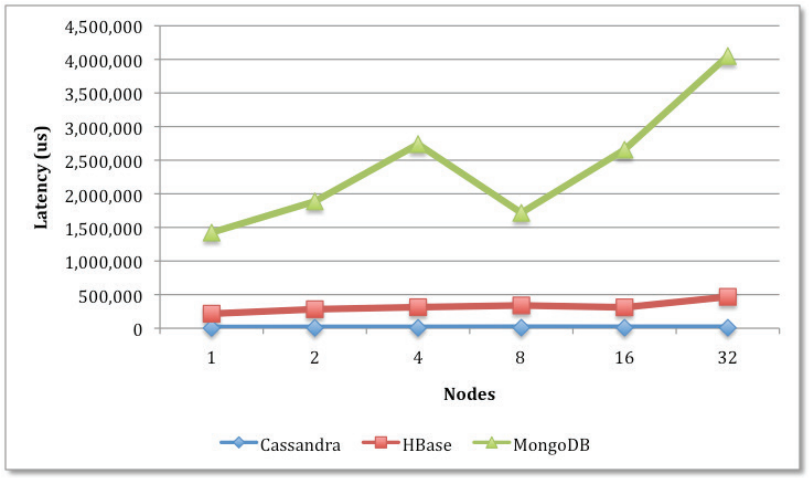
Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	225509.64	492450.15	499810.74	-118.37%	-121.64%
2	371531.12	638811.96	625352.17	-71.94%	-68.32%
4	340136.80	713152.11	719959.11	-109.67%	-111.67%
8	392649.61	766141.42	626442.52	-95.12%	-59.54%
16	395099.39	768299.09	751722.25	-94.46%	-90.26%
32	388945.97	1147158.86	585785.52	-194.94%	-50.61%

Insert latency across all workloads



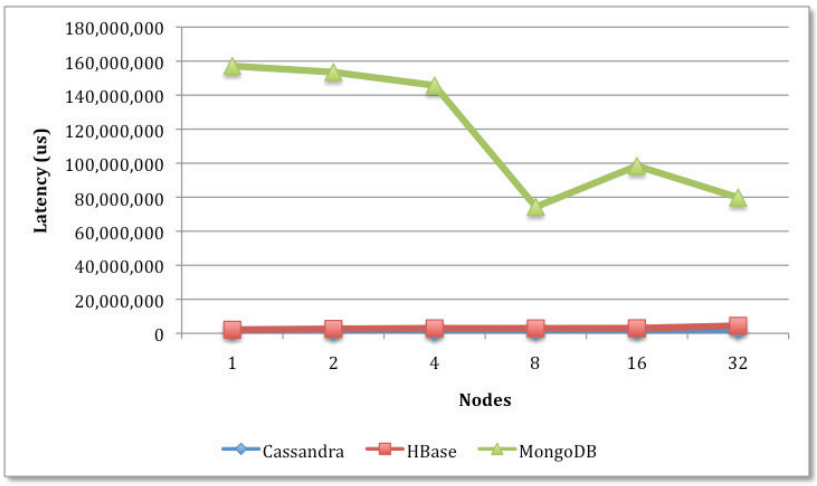
Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	3653.41	497232.17	287122.54	-13510.08%	-7759.03%
2	4187.51	592311.03	1453788.10	-14044.71%	-34617.24%
4	3654.28	707863.61	6481448.75	-19270.81%	-177265.96%
8	5432.62	700204.14	5489490.76	-12788.88%	-100946.84%
16	5007.98	723751.86	10566402.33	-14351.97%	-210891.30%
32	3138.56	1311862.19	18919252.94	-41698.22%	-602700.42%

Update latency across all workloads



Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	3292.73	216787.62	1423493.49	-6483.83%	-43131.41%
2	5244.74	282853.22	1887408.14	-5293.08%	-35886.69%
4	5477.54	313344.90	2738696.57	-5620.54%	-49898.66%
8	7875.85	339324.92	1716909.25	-4208.42%	-21699.67%
16	6975.09	310811.07	2656536.27	-4356.02%	-37986.05%
32	7052.18	467145.59	4050434.43	-6524.13%	-57335.21%

Scan latency across all workloads



Shards	Cassandra	HBase	MongoDB	%/HBase	%/MongoDB
1	1528571.77	2112239.43	157116882.28	-38.18%	-10178.67%
2	958200.98	2591140.03	153509366.73	-170.42%	-15920.58%
4	879313.90	2940485.45	145702505.42	-234.41%	-16470.02%
8	1398940.60	2907451.70	74329305.89	-107.83%	-5213.26%
16	1670090.64	2997596.75	98435275.64	-79.49%	-5794.01%
32	2113042.54	4403775.62	79787876.25	-108.41%	-3675.97%

Conclusion

As fast performance is key for nearly every data-driven system, IT professionals need to do their homework to ensure that that database they select is appropriate and targeted for their application use cases. The best way to do this is to conduct a formal POC in the environment in which the database will run and under the expected data and concurrent user workloads.

Benchmarks such as those contained in this document can be useful as well in that they give the developer and architect a good idea of what the core strengths and weaknesses that a proposed database possesses.

About DataStax

DataStax provides a massively scalable big data platform to run mission-critical business applications for some of the world's most innovative and data-intensive enterprises. Powered by the open source Apache Cassandra™ database, DataStax delivers a fully distributed, continuously available platform that is faster to deploy and less expensive to maintain than other database platforms.

DataStax has more than 250 customers including leaders such as Netflix, Rackspace, Pearson Education, and Constant Contact, and spans verticals including web, financial services, telecommunications, logistics, and government. Based in San Mateo, Calif., DataStax is backed by industry-leading investors including Lightspeed Venture Partners, Meritech Capital, and Crosslink Capital.

For more information, visit www.datastax.com.

About End Point

End Point provides development, consulting, and hosting services. The firm specializes in custom ecommerce, open source language and database support, and scalability assistance for clients large and small in many industries.

For more information, visit www.endpoint.com.

777 Mariners Island Blvd #510
San Mateo, CA 94404
650-389-6000