NoSQL Failover Characteristics: Aerospike, Cassandra, Couchbase, MongoDB

Denis Nelubin, Director of Technology, Thumbtack Technology
Ben Engber, CEO, Thumbtack Technology

Overview

Several weeks ago, we released a report entitled *Ultra-High Performance NoSQL Benchmarking: Analyzing Durability and Performance Tradeoffs*. The purpose of that study was to create a meaningful baseline for comparison across databases that take different approaches to availability and consistency. For this paper, we continued this study to examine one of the main reasons for using a NoSQL database — the ability to continue processing transactions in the face of hardware or other node failures.

In particular, we tried to answer how well the theoretical promises made by these platforms, i.e. that your system will continue to function normally in the face of such failures, performed in practice. We took the same suite of databases from the first study and performed a similar set of tests, but instead of focusing on raw performance numbers, we examined how failure and recovery events affected the system as a whole.

The motivation for this study was that raw performance numbers are, with some exceptions, not the primary motivation for picking a database platform. Horizontal scalability is the main design consideration when building internet-scale applications, and scaling the database layer is the most interesting part of the problem. Most discussions of NoSQL databases end up delving into long discussions of the CAP theorem and how each database deals with it, but since these systems are almost always running on a cluster, the real question is what consistency tradeoffs are needed to achieve a given level of performance. Or, in more practical terms, "How much data can I handle, how accurate is it, and what should I expect when something bad happens?" Here we tried to answer that last question in concrete terms based on systems commonly in production today.

The databases tested were Cassandra, Couchbase, Aerospike, and MongoDB. The hardware configurations used were identical those in the prior study.[1] We again used our customized version of the Yahoo Cloud Serving Benchmark (YCSB) as the basis for the test.

[1] For detailed information on the hardware and software configuration, please reference the original report at http://thumbtack.net/solutions/ThumbtackWhitePaper.html.

© 2013 Thumbtack Technology, Inc.

Test Description

The general approach to performing the tests was to bring each of these databases to a steady state of transactional load. We then brought one of the cluster nodes down in an unfriendly fashion and monitored the effects on latency and throughput, and how long it took for the database to once again hit a stable state. After 10 minutes of downtime, we would bring a node back into the cluster and perform system-specific recovery tasks. We then monitored the effect on performance and availability over the next 20 minutes.

We first ran this test at 50% of each database's maximum throughput (as measured in the prior study.) Given our four-node cluster, this ensured that even with one node down there was plenty of capacity to spare. An absolutely perfect system should show instantaneous failover and no impact whatsoever from node failures or recovery. In reality, of course, the failure needs to be detected, traffic rerouted, and ultimately data needs to be rereplicated when the node is reintroduced. Ideally, the database should strive for as close to zero impact as possible while supporting these features.

We also ran the same tests at 75% and 100% throughput on the cluster. At 75%, in theory there would be sufficient capacity to run even with one of the four nodes down, but with zero room to spare. This scenario represented an organization that invested in the minimal amount of hardware to support a one node loss. The 100% scenario represented the worst case scenario of a node failing when at capacity. We would expect performance to fall by at least 25% when we removed one of the nodes.

Other attributes we varied for the test were:

Replication Model:	Both synchronous and asynchronous replication
Durability Model:	Working set in RAM or written directly to disk
Workload:	Both read heavy and balanced workloads
Failure Type:	Simulated hardware failure versus network split brain[2]

In the interests of clarity, we did not include all the data we collected in our analysis. Many of the variables had little effect on the overall picture, other than changing the raw numbers in ways already discussed in the prior report.

The replication model did cause significant changes in how the cluster behaved under stress. We ran synchronous replication scenarios for Aerospike, Cassandra, and MongoDB, but were unable to get this to function for Couchbase. However, given our cluster size and the way Cassandra and MongoDB handle node failures, these databases were unable to perform

---

[2] `kill -9`, forced network failures, hardware power down, and other methods were tried and showed similar behavior. We opted to use `kill -9` as the baseline.

© 2013 Thumbtack Technology, Inc.

transactions while a node was down. This would not be true when using a larger replication factor, but was a necessary limitation to keep these results in the same baseline as our last report.

Client & Workload Description

As mentioned above, we ran the tests using two scenarios to represent different consistency levels of desired. The weak consistency scenario involved a data set that could fit entirely into RAM and was asynchronously replicated to a single replica. This was the classic eventually consistent model, and we expected it to provide the best results when dealing with node failures. The strong consistency scenario relied on synchronous replication, and used a larger data set.

Although we ran a broad swath of tests, we simplified the reporting of results for the sake of clarity. The baseline workload is described below.

Data Sets and Workloads

We used the same data sets and workloads as in the prior tests. To rehash:

Data Sets

Record description:	Each record consisted of 10 string fields, each 10 bytes long and with a 2-byte name
Record size:	120 bytes
Key description:	The key is the word “user” followed by a 64-bit Fowler-Noll-Vo hash[3] (in decimal notation)
Key size:	23 bytes
# of records (strong scenario):	200,000,000
# of records (weak scenario):	50,000,000

Workload

YCSB Distribution:	Zipfian
Balanced:	50% reads / 50% writes

We ran each test for 10 minutes to bring the database to a steady state, then took a node down, kept it down for 10 minutes, and then continued to run the test for an additional 20 minutes to examine recovery properties.

[3] http://en.wikipedia.org/wiki/Fowler_Noll_Vo_hash

© 2013 Thumbtack Technology, Inc.

Overview of Failover Behaviors

As with all measurements, the durability and consistency settings on the database have performance and reliability tradeoffs. The chart below shows some of the implications of the databases we tested and how they would typically be configured:

	Aerospike (async)	Aerospike (sync)	Cassandra (async)	Cassandra (sync)	Couchbase (async)	MongoDB (async)
Standard Replication Model	Asynchronous	Synchronous	Asynchronous	Synchronous	Asynchronous	Asynchronous
Durability	Asynchronous	Synchronous	Asynchronous	Asynchronous	Asynchronous	Asynchronous
Default sync batch	128kB per device	immediate	10 seconds	10 seconds	250k records	100 ms
Maximum write throughput per second[4]	230,000	95,000	22,000	22,000	270,000	24,000
Possible data loss on temporary node failure	large[5]	none	220,000 rows	none	large[6]	2400 rows
Consistency model[7]	Eventual	Immediate	Eventual	Immediate	Immediate*	Immediate*
Consistency on single node failure	Inconsistent	Consistent	Inconsistent	Consistent	Inconsistent	Inconsistent
Availability on single node failure / no quorum	Available	Available	Available	Unavailable[8]	Available	Available
Data loss on replica set failure[9]	25%	25%	25%	25%	25%	50%

All of these databases can be configured with other durability properties, for example MongoDB and Couchbase can be configured not to return success until data has been written to disk

---

[4] With these settings (from prior report with balanced workload)

[5] Synchronous disk writes were about 95,000 writes per second, so under high load much of the database could be stale

[6] Disk IO was measured at about 40,000 writes per second, so under high load much of the database could be stale

[7] Couchbase and MongoDB both offer immediate consistency by routing all requests to a master node.

[8] In our cluster. Technically, this should be written as “Availability when quorum not possible”, as it depends on the replication factor being used. With a replication factor of 3 or 4 instead of our 2, the system would be available when 1 replica is down but unavailable when 2 replicas are down.

[9] By “Data loss on replica set failure”, we mean the loss of the number of nodes equal to the replication factor. For MongoDB, this would mean losing all the nodes in a replica set.

© 2013 Thumbtack Technology, Inc.

and/or replicated, but we choose the ones that worked well in our testing and would be used in most production environments.

Results

Trying to quantify what happens during failover is complex and context-dependent, so before presenting the raw numbers, we give an overview of what happens during failover for each of these systems. We then present graphs of the databases performance over time in the face of cluster failures, and then attempt to quantify some of the behaviors we witnessed.

For clarity, in this section we primarily show the behaviors for databases operating with a working set that fits into RAM. We also tested with a larger data set that went to disk. Those results were slower but similar in content, though with more noise that makes reading some of the graphs difficult. We felt the RAM dataset is better for illustrating the failover behavior we experienced.[10]

	Aerospike (async)	Aerospike (sync)	Cassandra (async)	Cassandra (sync)	Couchbase (async)	MongoDB (async)
Original Throughput	300,000	150,000	27,500	30,000	375,000	33,750
Original Replication	100%	100%	99%	104%	100%	100%
Downtime (ms)	3,200	1,600	6,300	∞	2,400*	4,250
Recovery time (ms)	4,500	900	27,000	N/A	5,000	600
Node Down Throughput	300,000	149,200	22,000	0	362,000	31,200
Node Down Replication	52%	52%	N/A	54%	50%	50%
Time to stabilize on node up (ms)	small	3,300	small	small	small	31,100
Final Throughput	300,000	88,300	21,300†	17,500†	362,000	31,200
Final Replication	100%	100%	101%	108%	76%	100%

† Depends on driver being used. Newer drivers like Hector restore to 100% throughput

* Assuming perfect monitoring scripts

[10] Our earlier report provides a detailed explanation of how these databases perform with a disk-based data set, for those who are interested.

© 2013 Thumbtack Technology, Inc.

Cluster Behavior Over Time

Below are some graphs that represent how the databases behaved over the full course of the cluster disruption and recovery. For the sake of clarity, we do not show every test scenario, but merely some representative cases that illustrate the behavior we saw.

Interpreting performance over time

The graphs below illustrate different behaviors of the databases through the lifecycles of some representative tests.

The applications behaved similarly under the default case of 75% throughput using asynchronous replication and a RAM-based data set. In all the cases, there was a brief period of cluster downtime when a node went down, followed by continued throughput at or near the original level. When the node rejoined the cluster, there was another brief period of downtime followed by a throughput quickly being restored to the original level. The main difference between databases was the level of volatility in latencies during major events. Aerospike maintained the most consistent performance throughout. Cassandra showed increased fluctuations while the node was down, and MongoDB became significantly more volatile as the node rejoined the cluster. Couchbase had the peculiar characteristic of decreased volatility while the node was down, presumably because of reduced replication.

Under 100% throughput, Aerospike, Cassandra, and Couchbase each saw capacity drop by 25% when a node went down, exactly as one would expect when losing one of four machines. MongoDB showed no change; again this is what is expected given their replica set topology (the number of nodes servicing requests is unchanged when a slave takes over for the downed master.) When the node was brought back and rejoined the cluster, all the databases recovered to near full throughput, though Couchbase took some time to do so. (In the picture below, Cassandra throughput did not recover, but this is an artifact of the client driver's reconnect settings and does not represent database behavior.)

When running the tests with synchronous replication and using disk-based persistence, some interesting trends are visible. Given a replication factor of 2, only Aerospike was able to keep servicing synchronous requests on a node down event — it simply chose a new node for writing replicas on incoming write operations. Both Cassandra and MongoDB simply failed for updates that would have involved the missing replica. This resulted in downtime for the duration of the node down event, but a rapid recovery to full capacity as soon as the missing node became active again.[11] A corollary for Aerospike is that there is substantial replication effect when the node comes back and more current data is migrated to it, which can easily be seen in the graph. As in our prior tests, we were unable to get Couchbase to function in a purely synchronous manner.

---

[11] If the replication factor were three, the writes should succeed. A more complete accounting of this will be presented in a future report.

© 2013 Thumbtack Technology, Inc.

75% load, asynchronous replication, RAM-based data set

Figure 1a: Aerospike

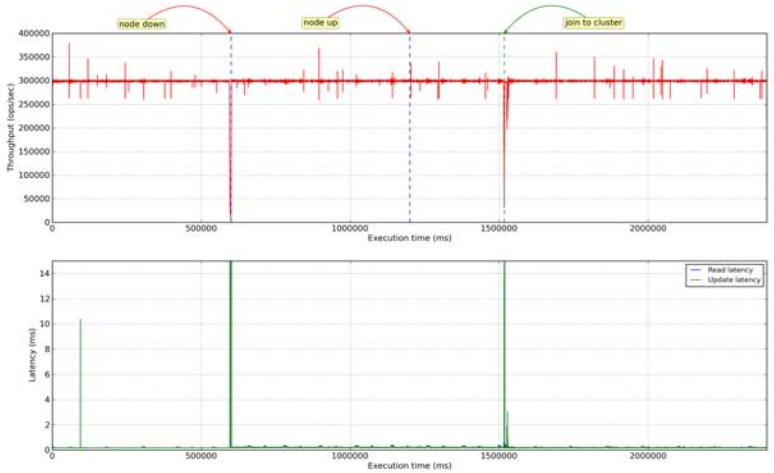


Figure 1b: Cassandra

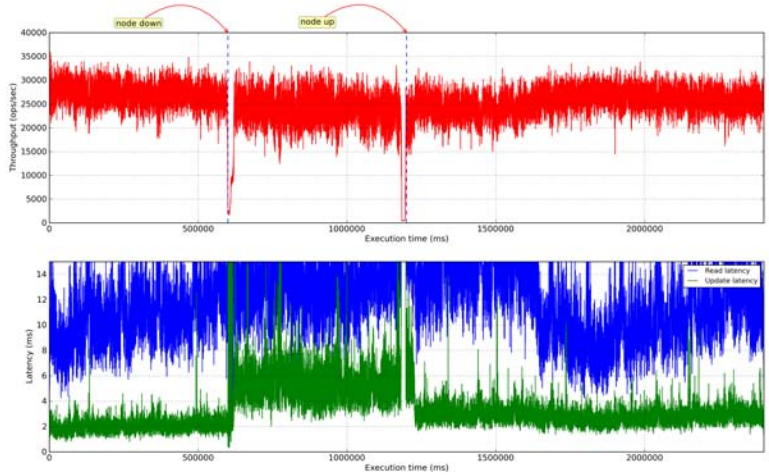


Figure 1c: Couchbase

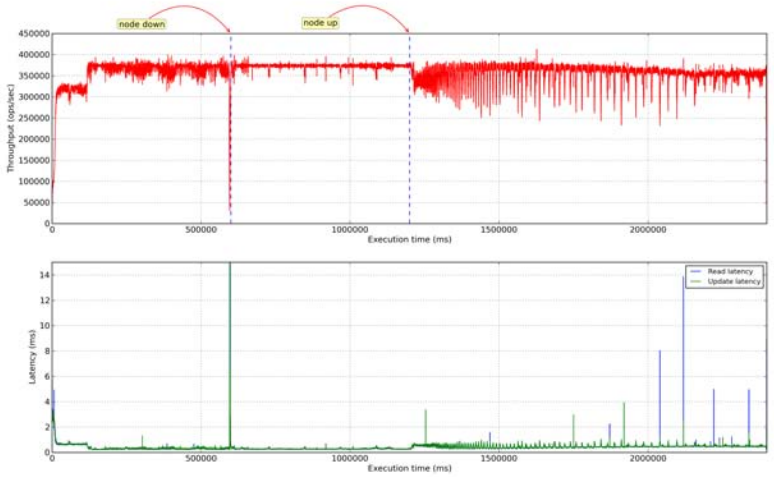
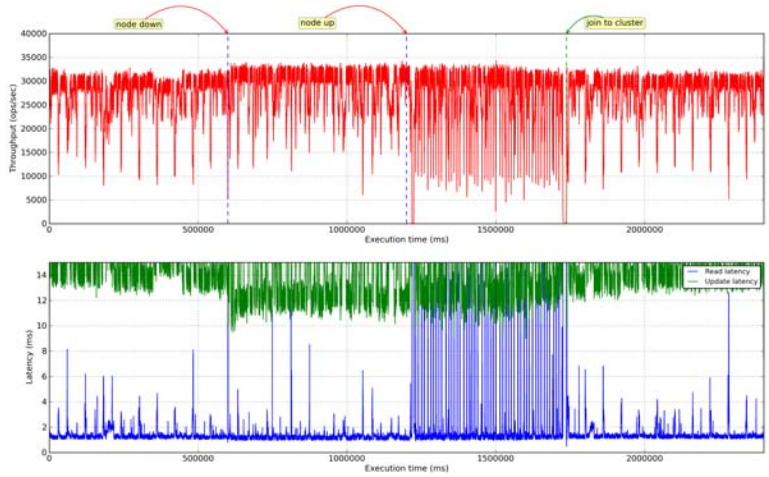


Figure 1d: MongoDB



© 2013 Thumbtack Technology, Inc.

100% load, asynchronous replication, RAM-based data set

Figure 2a: Aerospike

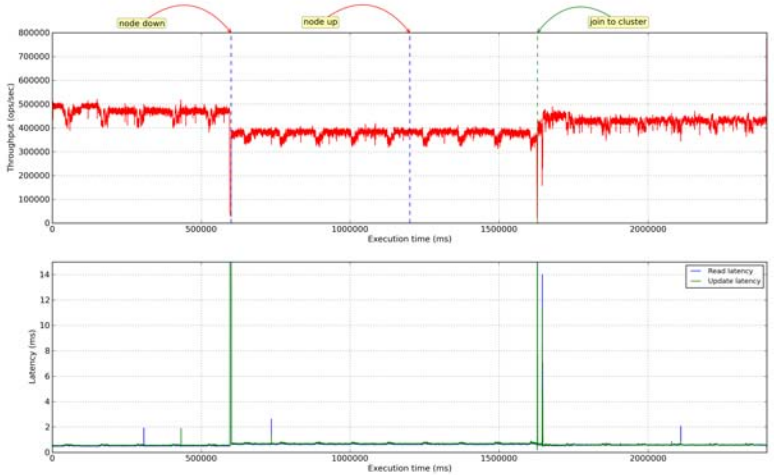


Figure 2b: Cassandra

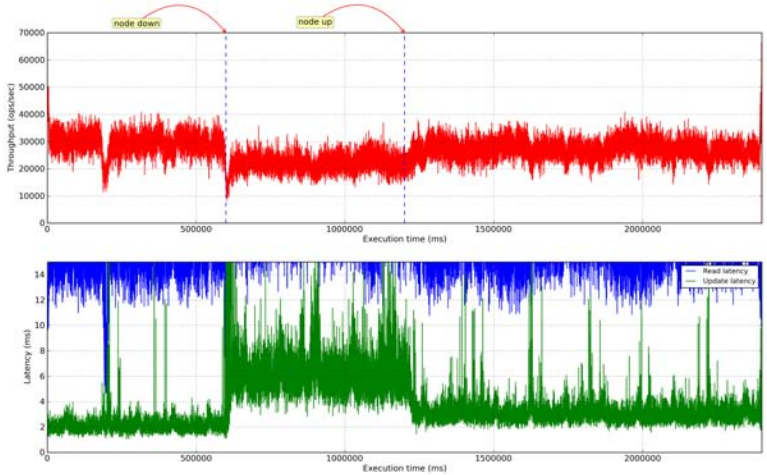


Figure 2c: Couchbase

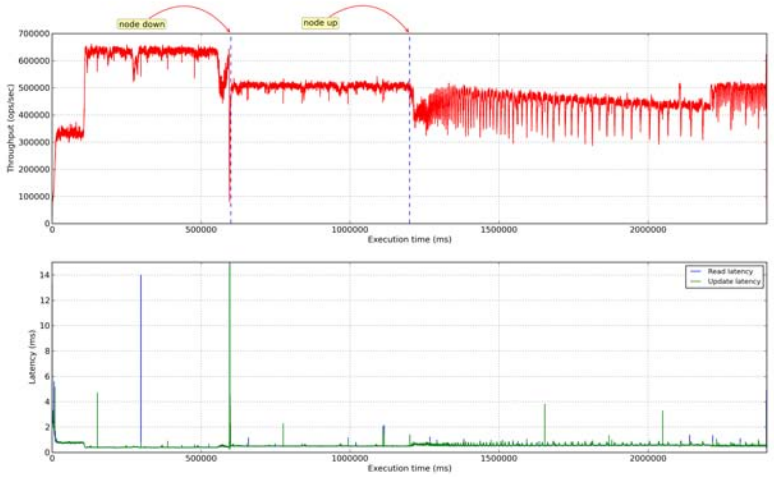
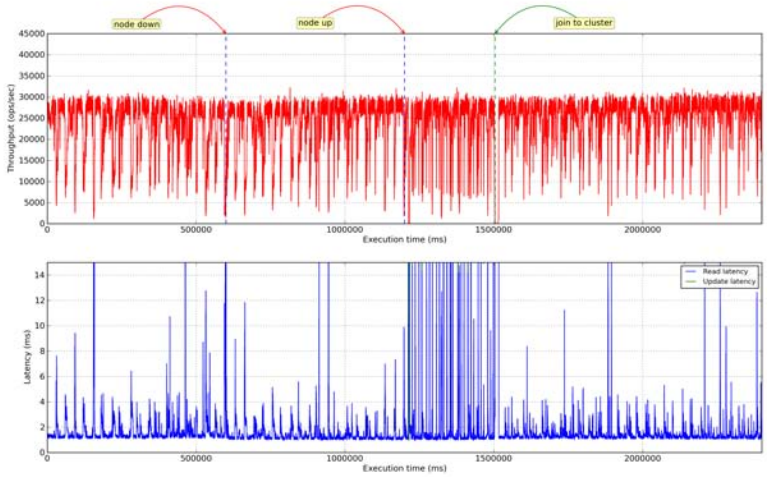


Figure 2d: MongoDB



© 2013 Thumbtack Technology, Inc.

75% load, synchronous replication, SSD-based data set

Figure 3a: Aerospike

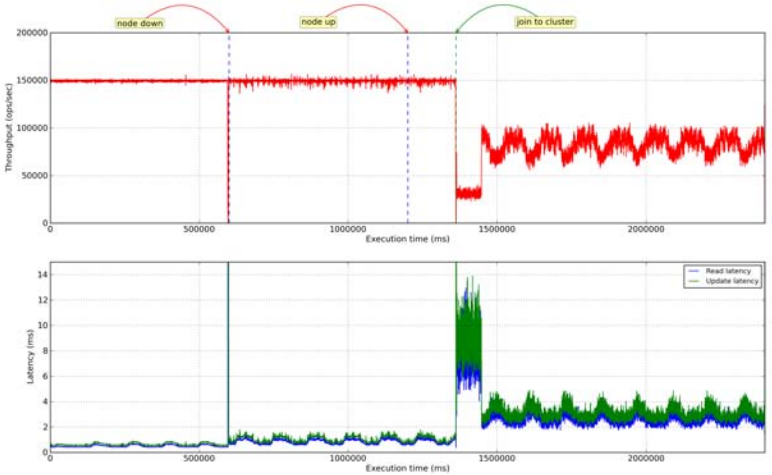


Figure 3b: Cassandra

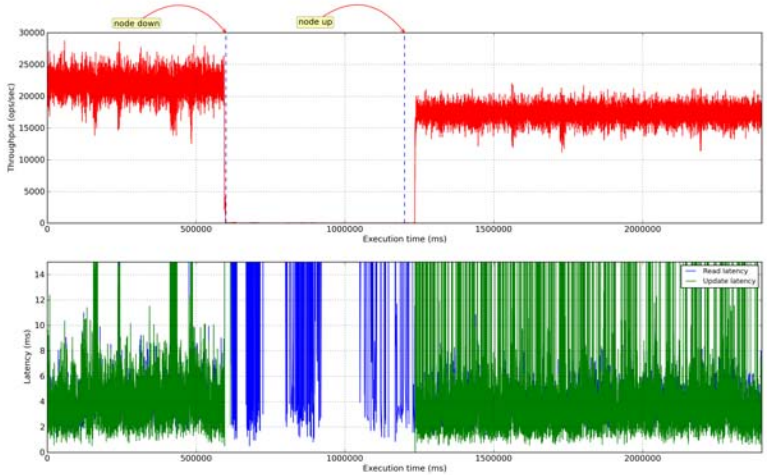
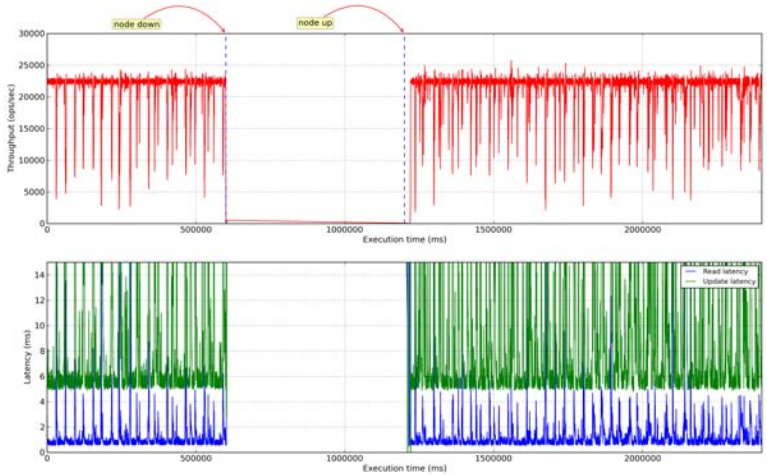


Figure 3c: Couchbase (N/A)

Figure 3d: MongoDB



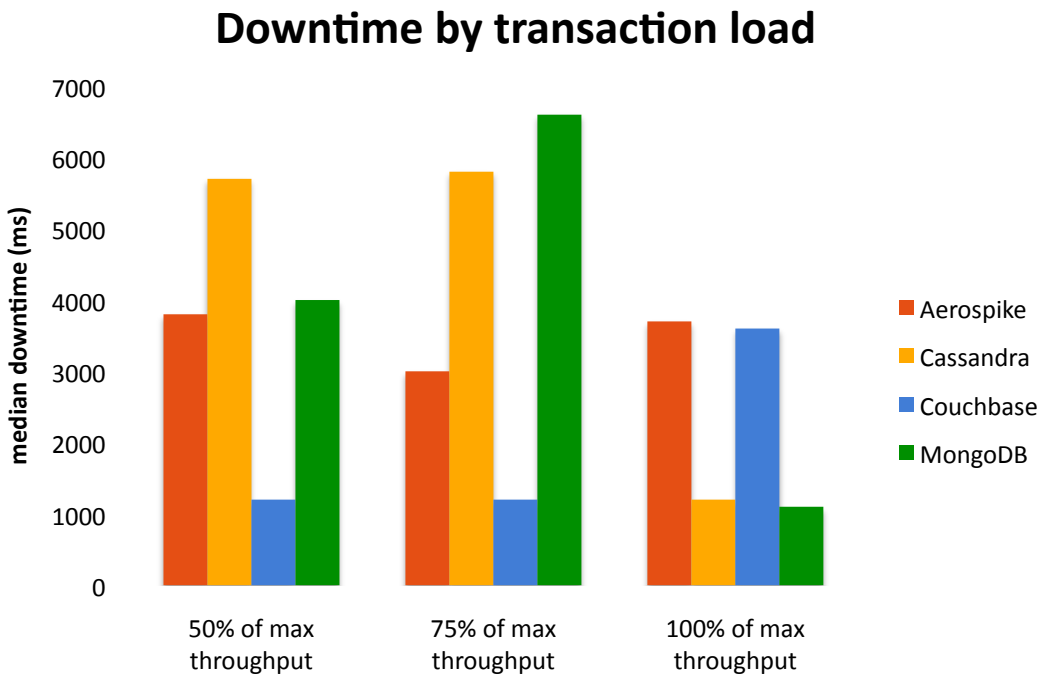
© 2013 Thumbtack Technology, Inc.

Node Down Behavior

We measured how long it takes for the database cluster to become responsive again (which we defined as handling at least 10% of prior throughput) during a node down event. For this test, the databases were running in an asynchronous mode. We examined the amount of time the cluster was unavailable and the subsequent effect on performance with the node down.

All the databases performed quite well in this scenario. MongoDB, Couchbase, and Aerospike all became available within 5 seconds of the event, while Cassandra took up to 20 seconds under load. In the case of both MongoDB and Couchbase, the recovery time was close to immediate (but see note below).

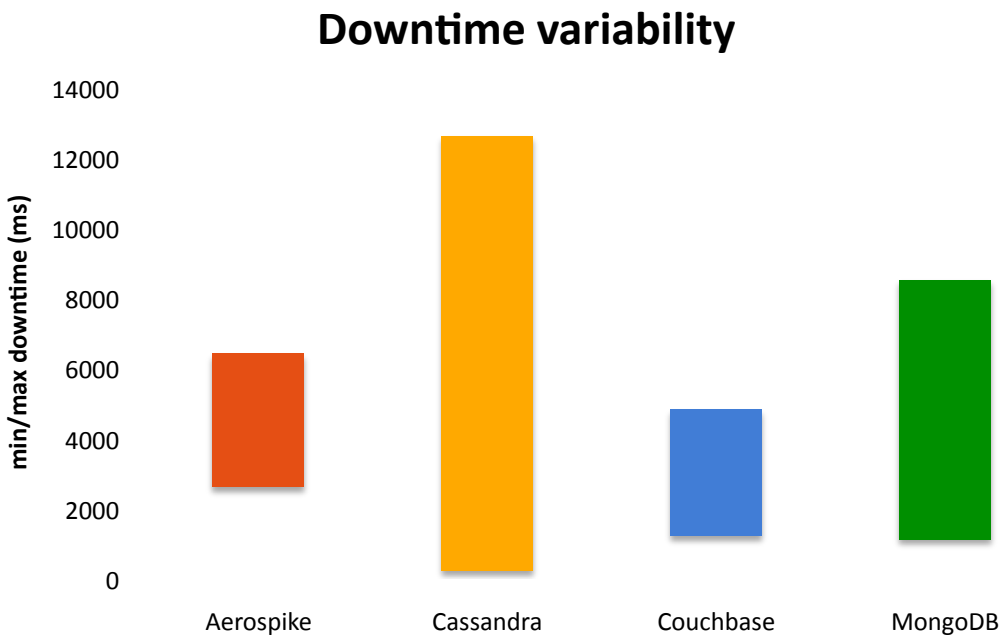
Figure 4a: Downtime, asynchronous replication, RAM-based data set[12]



[12] We do not include a graph of downtime in synchronous mode. As discussed earlier, Cassandra will not function in synchronous mode with a replication factor of 2 (though it will with larger replication factors), and Couchbase and MongoDB are not designed with synchronous replication in mind. For Aerospike, synchronous replication worked as advertised and had similar downtime numbers.

© 2013 Thumbtack Technology, Inc.

Figure 4b: Downtime, variability by database

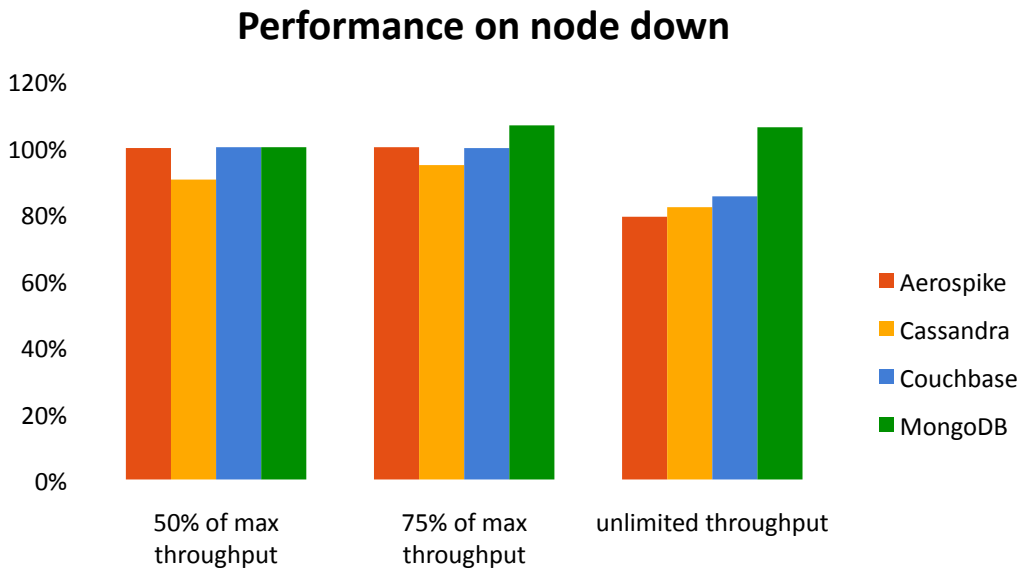


Some caveats should be made in interpreting these results. First of all, we use a manual failover for Couchbase. Couchbase's auto-failover has a minimum value of 30 seconds, and when using it we saw downtimes of 30-45 seconds. In contrast to the other products tested, Couchbase recommends doing independent monitoring and failover of the cluster, and so we assumed a near-perfect system that detected failures within 1 second. In reality, it would not be realistic to assume a true failure based on one second of inactivity. What we can conclude from this test is that when using outside monitoring, Couchbase can recover quickly if the monitors and recovery scripts are reliable.

In short, we felt all of these products performed admirably in the face of node failures, given that the frequency of such events are quite small, and all the times listed here are probably within the level of noise in monitoring the system in general.

© 2013 Thumbtack Technology, Inc.

Figure 5: Relative speed on node down (asynchronous replication, RAM-based data set)



Once the cluster becomes available after a failure, the performance remained unaffected for the 50% and 75% scenarios, exactly as we expected. For the 100% load scenario, the performance degraded to approximately 75% as expected, with the exception of MongoDB, which continued to perform at full speed since the formerly unused secondary nodes kicked in to continue performance. (In some tests, the speed actually increased, which we chalked up to the fact that replication overhead was no longer needed.)

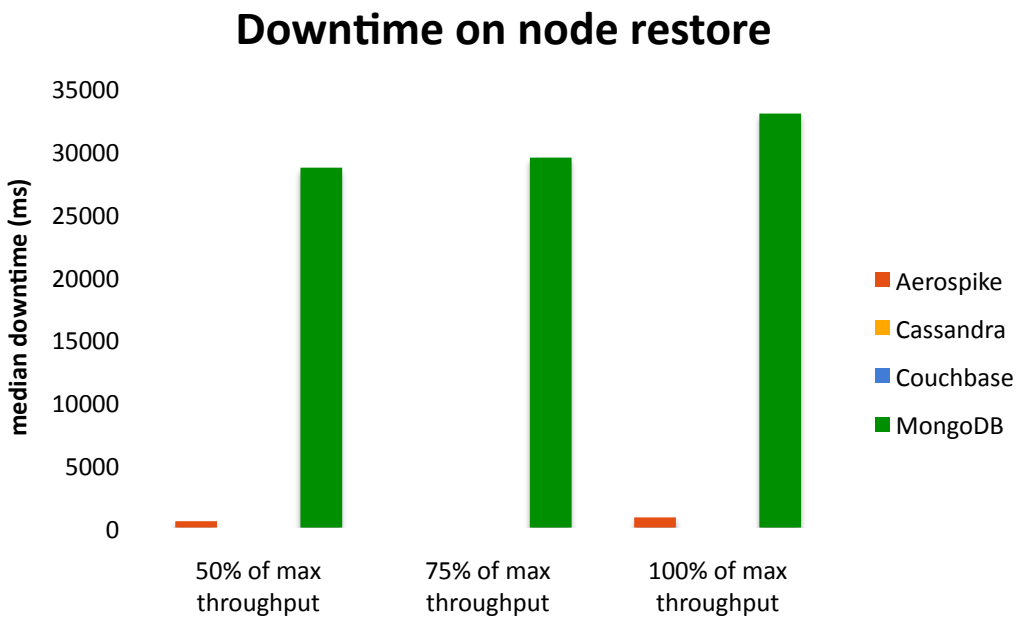
Before the failure, latency for all the systems is extremely low, but after the node fails all the systems slow down considerably.

Node Recovery Results

In general, restoring a cluster is a more expensive operation than losing a node, since the database must first detect and resolve any conflicting updates and then replicate over any data on the new node that might be stale. In our tests, all the databases were able to perform this operation quite well with little downtime or impact on throughput.

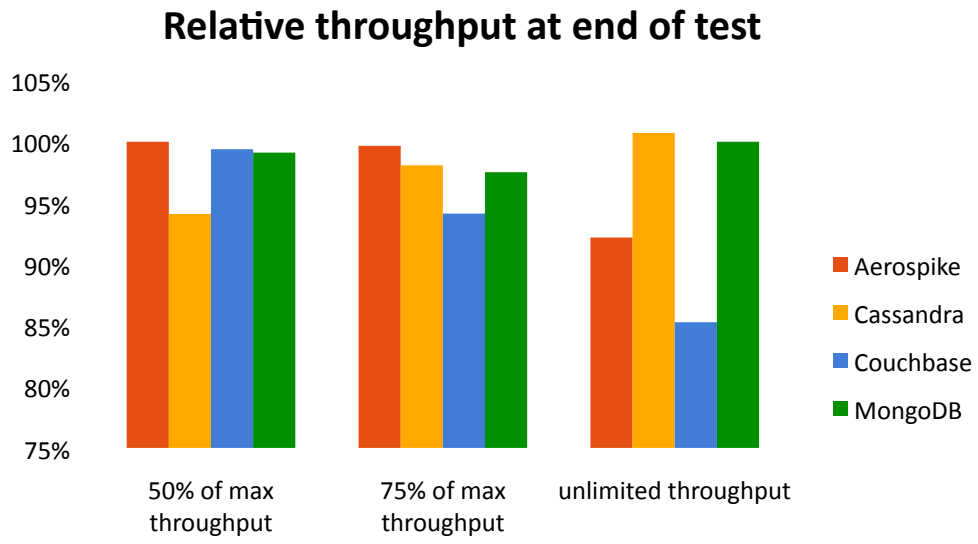
© 2013 Thumbtack Technology, Inc.

Figure 6: Downtime during node join (asynchronous replication, RAM-based data set)



All the databases started servicing requests almost immediately, except for MongoDB which had about 30 seconds of downtime when rejoining the cluster.

Figure 7: Relative performance after node joins (asynchronous replication, RAM-based data set)



As is clear from the 100% load scenario, throughput on the systems did not recover immediately once the cluster is repaired (in the case of MongoDB, since the throughput never dropped, it did not need to recover.) Once the new nodes were brought to a fully consistent state through

© 2013 Thumbtack Technology, Inc.

replication, performance recovered completely. The length of time it took for this replication to complete is not a fair metric, since the amount of data being pushed through the systems varied dramatically by database. We can say that for all databases, throughput eventually recovered to starting values.

Conclusions

The central conclusion we made is that these products tend to perform failover and recovery as expected, with varying levels of performance fluctuations during the tests. Even under heavy write load, the interruptions in service were limited to 30 seconds or less. This was repeated in numerous tests, using different methods of disrupting the cluster, and using different kinds of workloads, storage models, and replication settings. The truth is that all these databases performed were able to detect and automatically handle failure conditions and resume serving requests quickly enough to not make this the primary concern.

The behavior of the databases as they handle the conditions is interesting. Of the four databases we tested, only Aerospike was able to function in synchronous mode with a replication factor of two. With a larger cluster and larger replication factor this is no longer true. However, it is a significant advantage that Aerospike is able to function reliably on a smaller amount of hardware while still maintaining true consistency.

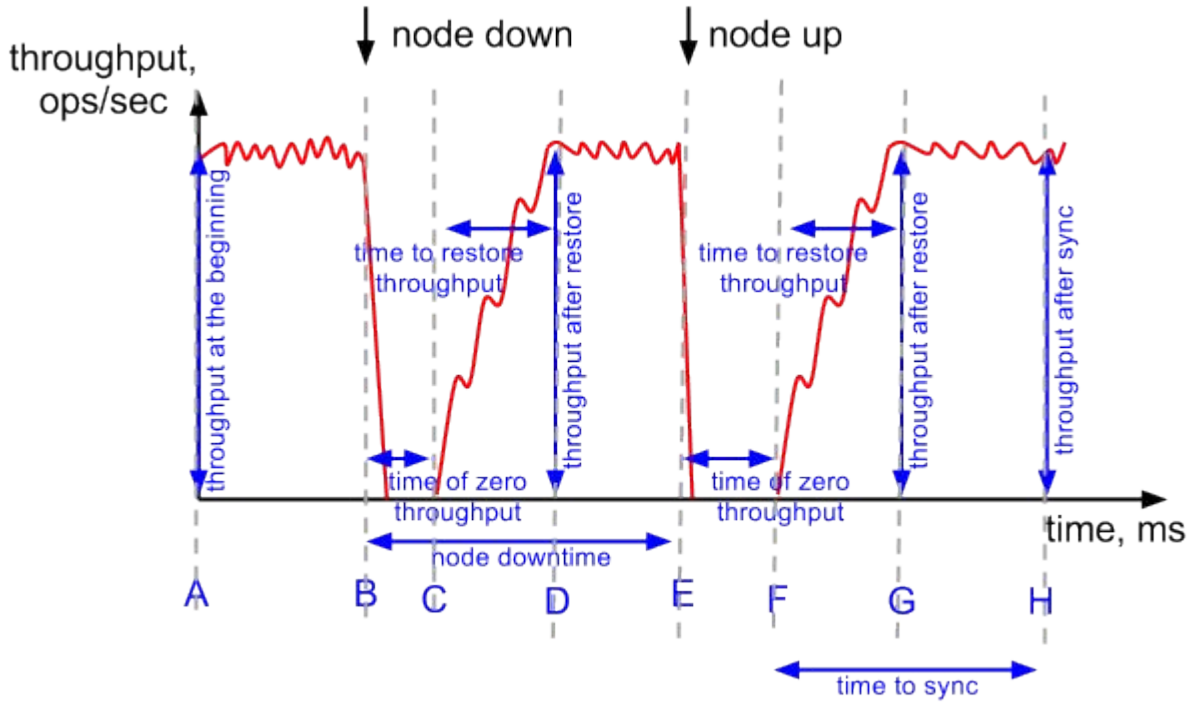
As discussed in the beginning of our results section, one of the major disadvantages in running in asynchronous mode is the potential for data loss on node outages. This can mean data inconsistency in the case of a transient failure such as a network outage, or complete data loss in the case of of a disk failure. Attempting to quantify this in a reproducible way was quite difficult, and the tradeoff between performance and replication speed is tunable on some of these systems. We did offer a theoretical amount of data loss based on the ways these databases sync to disk.

During our tests we did discover some bugs in some of the products, all of which were fairly easily worked around with relatively minor configuration changes. Such is the nature of testing emerging technologies. Once those issues were accounted for, decisions between which system to choose for failover should be made based on are more on decisions based on how much data loss is acceptable (if any), and how much to invest in hardware versus software.

Lastly, we provide the obligatory caveat that no matter how much one tries to break something in the lab, once it hits production there is always a new way. We can't say these systems won't break in production, but we can say they all seemed to function as advertised in our tests with little user interaction.

© 2013 Thumbtack Technology, Inc.

Appendix A: Detailed Test List



Load 50 Million (or 200 Million) records to 4 node cluster

Load the complete dataset into each database. This was done once and then reused for each of the following tests. In cases when waiting for rebalancing to be completed took longer than erasing and reloading data, we simply rebuilt the database.

The charts in the paper are all based on the 50 million record data set. The 200 million record data set was used to force disk access. The results were slower but not appreciably different in meaning.

General Failover Test

We ran the YCSB Workload A (50% reads, 50% updates) on the cluster while limiting the throughput to 50% maximum throughput the database can handle (known from our prior study). After 10 minutes we would terminate a database on one node using the `kill -9` command. After 10 more minutes we would restart the process and rejoin the node to the cluster. We would then wait 20 minutes to observe the behavior as the node joined the cluster.

On a node failure:

- For Aerospike, Cassandra, and MongoDB we did nothing and let the built-in auto-recovery handle the situation.

© 2013 Thumbtack Technology, Inc.

- For Couchbase, we used two methods:
    - The built-in auto-recovery, which takes 30 to 45 seconds to take effect.
    - A manual process:
        - Wait 1 second to simulate delay of automated monitoring software
        - Run the `couchbase-cli failover` command.
        - Wait 3 seconds (best value, by trial and error).
        - Run the `couchbase-cli rebalance` command.

To rejoin the cluster, we would use the following commands:

- Aerospike: /etc/init.d/citrusleaf start
- Cassandra: /opt/cassandra/bin/cassandra
- Couchbase: /etc/init.d/couchbase-server start; sleep 7; couchbase-cli server-add; sleep 3; couchbase-cli rebalance;
- MongoDB: /opt/mongodb/bin/mongod with all usual necessary parameters

Test Variations

We reran the above tests by varying different parameters:

Throughput

We ran the tests at three different load capacities.

- 50% — representing having plenty of hardware to spare
- 75% — representing the theoretical maximum that could be handled by the cluster with a node down
- 100% — representing what would happen under extreme stress

Replication

We ran the tests using both synchronous and asynchronous replication for each database. The way this is achieved is database-dependent and described in the original report. Couchbase did not work reliably under synchronous replication, regardless of the size of the data set (it is not the standard way Couchbase is used).

Data Set

We used both a data set of 50 million records to represent a working set that fits in RAM, as well as a data set of 200 million records backed by SSD.

© 2013 Thumbtack Technology, Inc.

Workload

We ran a workload of 50% reads and 50% writes, and also with 95% reads and 5% writes.

Node Failure Type

We tried two types of node failures in our tests:

- Hardware failure: Simulated by `kill -9` on the server process
- Network / split brain: Simulated by raising a firewall between nodes

Metrics

We track the amount of time the cluster is unavailable by measuring the amount of time total throughput remains less than 10% of the known capacity.

Replication statistics, when gathered, were determined by using the following commands:

- Aerospike: clmonitor -e info
- Cassandra: nodetool cfstats
- Couchbase: number of replica items was monitored through web console
- MongoDB: rs.status() to see which node is up and down, db.usertable.count() to check number of documents in a replica-set

Other measurements were performed directly.

Run failover test, Workload A, 75% of max throughput

The same as the test above, but the throughput is limited to 75% of known maximum throughput of the database.

Run failover test, Workload A, 100% of max throughput

The same as the test above, but the throughput is not limited.

Resetting Tests

After a test is completed, but before we began another, we performed the following actions:

- Shut down all DB instances.
- Ensure all server processes are not running.
- Leave data on disk.

© 2013 Thumbtack Technology, Inc.

Appendix B: Hardware and Software

Database Servers

We will run the tests on four server machines. Each machine has the following specs:

CPU:	8 x Intel(R) Xeon(R) CPU E5-2665 0 @ 2.40GHz
RAM:	31 GB[13]
SSD:	4 x INTEL SSDSA2CW120G3, 120 GB full capacity (94 GB over-provisioned)
HDD:	ST500NM0011, 500 GB, SATA III, 7200 RPM
Network:	1Gbps ethernet
OS:	Ubuntu Server 12.04.1 64-bit (Linux kernel v.3.2.0)
JDK:	Oracle JDK 7u9

Client Machines

We used eight client machines to generate load to the database with YCSB. Each had the following specs:

CPU:	4 x Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz
RAM:	3.7 GB
HDD:	ST500DM002-1BD142, 500 GB, SATA III, 7200 RPM
OS:	Ubuntu Server 12.04.1 64-bit (Linux kernel v.3.2.0)
JDK:	Oracle JDK 7u9

For further information on how these machines were configured, please refer to Appendices A and B in our prior report.

Database Software

- Aerospike 2.6.0 (free community edition)
- Couchbase 2.0.0
- Cassandra 1.1.7
- MongoDB 2.2.2

For detailed database configuration information, please refer to Appendix C of the prior report.

[13] 32 GB of RAM, 1 GB of which is reserved for integrated video

© 2013 Thumbtack Technology, Inc.

Client Software

- Thumbtack's own customized version of YCSB, available from https://github.com/thumbtack-technology/ycsb.

For details of the changes made to YCSB, please refer to Appendix E of the prior report. Minor additional error logging changes were made for this follow up study, primarily to deal with MongoDB and Cassandra errors we encountered.

© 2013 Thumbtack Technology, Inc.