A Vendor-independent Comparison of NoSQL Databases: Cassandra, HBase, MongoDB, Riak

By Sergey Bushik, Senior R&D Engineer at Altoros

"The more alternatives, the more difficult the choice."
Abbe' D'Allanival

Contents

1. Introduction

In 2010, when the world became enchanted by the capabilities of cloud systems and new databases designed to serve them, a group of researchers from Yahoo decided to look into NoSQL. They developed the YCSB framework to assess the performance of new tools and find the best cases for their use. The results were published in the paper, *"Benchmarking Cloud Serving Systems with YCSB."*

The Yahoo guys did a great job, but like any paper, it could not include everything:

- The research did not provide all the information we needed for our own analysis.
- Though Cassandra, HBase, Yahoo's PNUTS, and a simple sharded MySQL implementation were analyzed, some of the databases we often work with were not covered.
- Yahoo used high-performance hardware, while it would be more useful for most companies to see how these databases perform on average hardware.

As R&D engineers at Altoros Systems, Inc., a big data specialist, we were inspired by Yahoo's endeavors and decided to add some effort of our own. This article is our vendor-independent analysis of NoSQL databases, based on performance measured under different system workloads.

2. What Makes This Research Unique?

Often referred to as NoSQL, non-relational databases feature elasticity and scalability in combination with a capability to store big data and work with cloud computing systems, all of which make them extremely popular. NoSQL data management systems are inherently schema-free (with no obsessive complexity and a flexible data model) and eventually consistent (complying with BASE rather than ACID). They have a simple API, serve huge amounts of data and provide high throughput.

In 2012, the number of NoSQL products reached 120+ and the figure is still growing. That variety makes it difficult to select the best tool for a particular case. Database vendors usually measure productivity of their products with custom hardware and software settings designed to demonstrate the advantages of their solutions. We wanted to do independent and unbiased research to complement the work done by the folks at Yahoo.

Using Amazon virtual machines to ensure verifiable results and research transparency (which also helped minimize errors due to hardware differences), we have analyzed and evaluated the following NoSQL solutions:

- Cassandra, a column family store
- HBase (column-oriented, too)
- MongoDB, a document-oriented database
- Riak, a key-value store

We also tested MySQL Cluster and sharded MySQL, taking them as benchmarks.

After some of the results had been presented to the public, some observers said MongoDB should not be compared to other NoSQL databases because it is more targeted at working with memory directly. We certainly understand this, but the aim of this investigation is to determine the best use cases for different NoSQL products. Therefore, the databases were tested under the same conditions, regardless of their specifics.

Tools, Libraries, and Methods

For benchmarking, we used Yahoo Cloud Serving Benchmark, which consists of the following components:

- a framework with a workload generator
- a set of workload scenarios

We have measured database performance under certain types of workloads. A workload was defined by different distributions assigned to the two main choices:

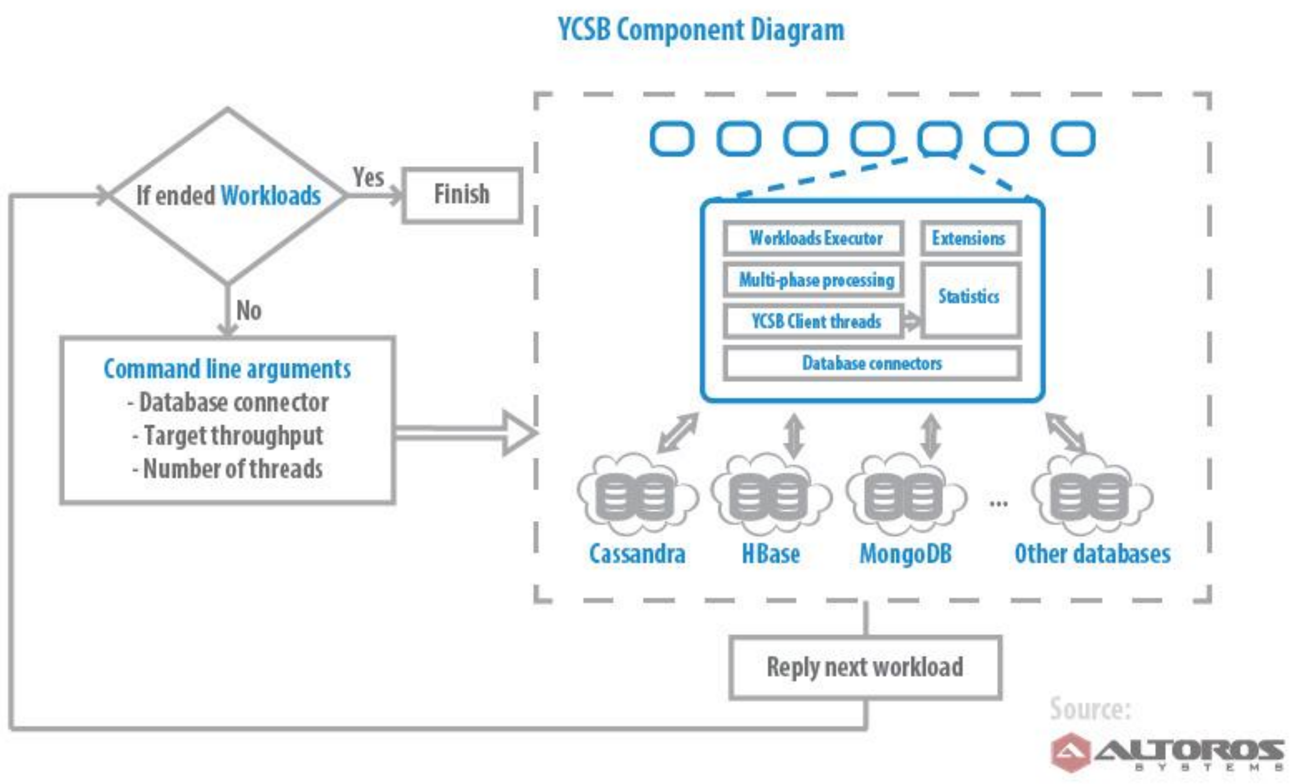
- which operation to perform
- which record to read or write

Operations against a data store were randomly selected and could be of the following types:

- *Insert*: Inserts a new record.
- *Update*: Updates a record by replacing the value of one field.
- *Read*: Reads a record, either one randomly selected field, or all fields.
- *Scan*: Scans records in order, starting at a randomly selected record key. The number of records to scan is also selected randomly from the range between 1 and 100.

Each workload was targeted at a table of 100,000,000 records; each record was 1,000 bytes in size and contained 10 fields. A primary key identified each record, which was a string, such as “user234123.” Each field was named field0, field1, and so on. The values in each field were random strings of ASCII characters, 100 bytes each.

Database performance was defined by the speed at which a database computed basic operations. A basic operation is an action performed by the workload executor, which drives multiple client threads. Each thread executes a sequential series of operations by making calls to the database interface layer both to load the database (the load phase) and to execute the workload (the transaction phase). The threads throttle the rate at which they generate requests, so that we may directly control the offered load against the database. In addition, the threads measure the latency and achieved throughput of their operations and report these measurements to the statistics module.



The performance of the system was evaluated under different workloads:

Workload A: Update heavily

Workload B: Read mostly

Workload C: Read only

Workload D: Read latest

Workload E: Scan short ranges

Workload F: Read-modify-write

Workload G: Write heavily

Each workload was defined by:

1) The number of records manipulated (read or written)
2) The number of columns per each record
3) The total size of a record or the size of each column
4) The number of threads used to load the system

This research also specifies configuration settings for each type of the workloads. We used the following default settings:

1) 100,000,000 records manipulated
2) The total size of a record equal to 1 Kb
3) 10 fields of 100 bytes each per record
4) Multithreaded communications with the system (100 threads)

3. Testing Environment

To provide verifiable results, benchmarking was performed on Amazon Elastic Compute Cloud instances. Yahoo Cloud Serving Benchmark Client was deployed on one Amazon Large Instance:

- 7.5 GB of memory
- four EC2 Compute Units (two virtual cores with two EC2 Compute Units each)
- 850 GB of instance storage
- 64-bit platform
- high I/O performance
- EBS-Optimized (500 Mbps)
- API name: m1.large

Each of the NoSQL databases was deployed on a four-node cluster in the same geographical region on Amazon Extra Large Instances:

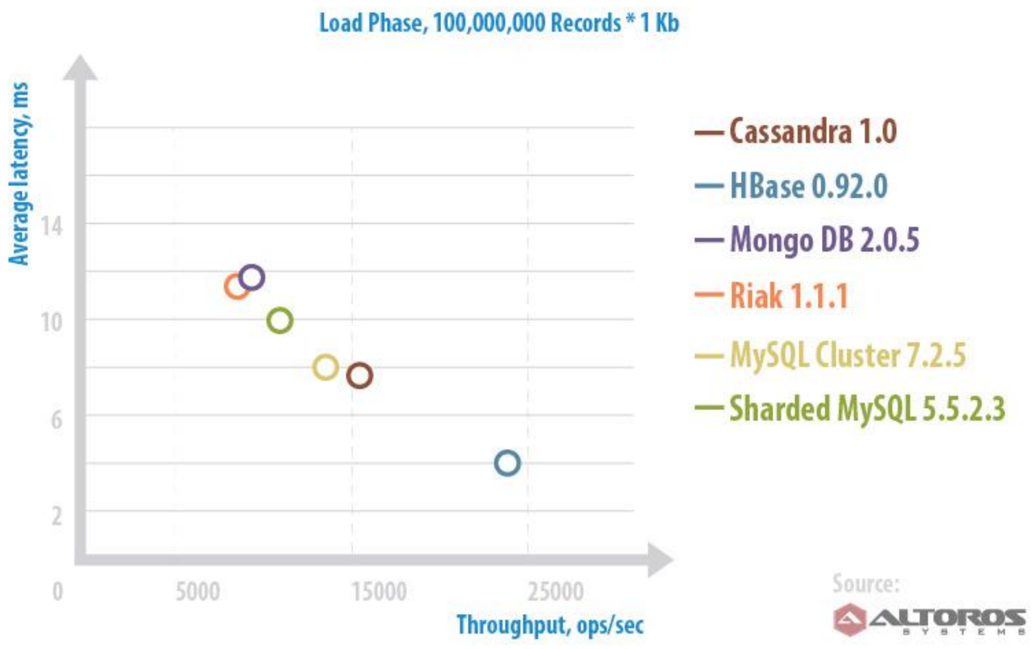
- 15 GB of memory
- eight EC2 Compute Units (four virtual cores with two EC2 Compute Units each)
- 1,690 GB of instance storage
- 64-bit platform
- high I/O performance
- EBS-Optimized (1000 Mbps)
- API name: m1.xlarge

Amazon is often blamed for its high I/O wait time and comparatively slow EBS performance. To mitigate these drawbacks, EBS disks had been assembled in a RAID0 array with stripping and after that they were able to provide up to two times higher performance.

4. The Results

When we started our research into NoSQL databases, we wanted to get unbiased results that would show which solution is best suitable for each particular task. That is why we decided to test performance of each database under different types of loads and let the users decide what product better suits their needs.

We started with measuring the load phase, during which 100 million records, each containing 10 fields of 100 randomly generated bytes, were imported to a four-node cluster.

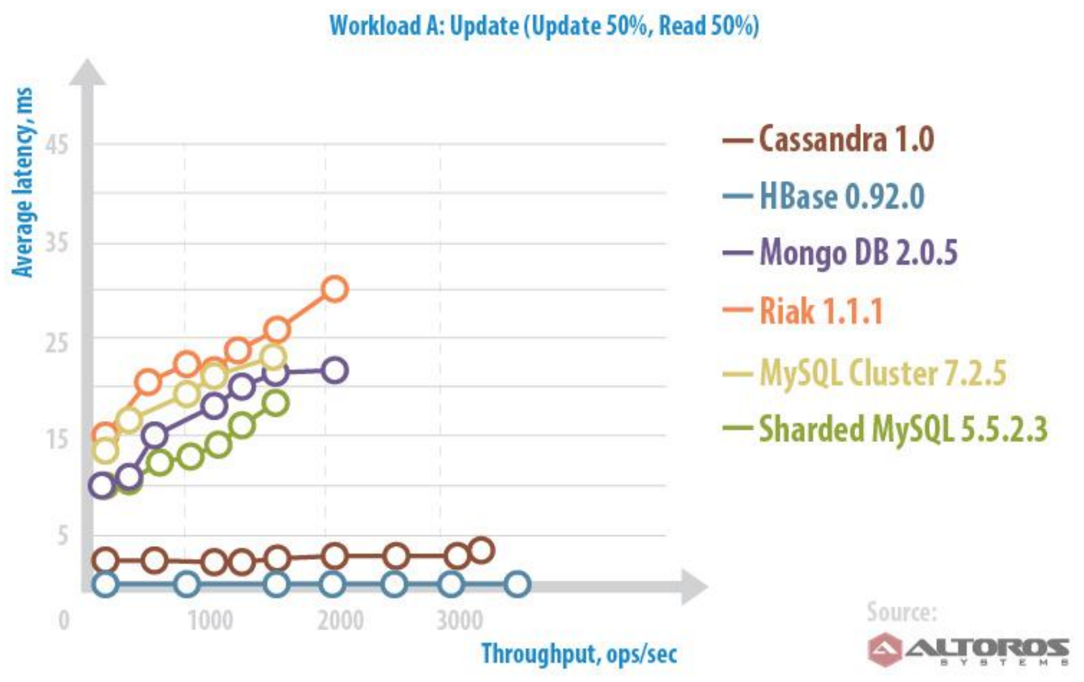


HBase demonstrated by far the best writing speed. With pre-created regions and deferred log flush enabled, it reached 40K ops/sec. Cassandra also showed great performance during the loading phase with around 15K ops/sec. The data is first saved to the commit log, using the append method, which is a fast operation. Then it is written to a per-column family memory store called a Memtable. Once the Memtable becomes full, the data is saved to disk as an SSTable. In the “just in-memory” mode, MySQL Cluster could show much better results, by the way.

*** Workload A: Update-heavily mode.** Workload A is an update-heavily scenario that simulates the database work, during which typical actions of an e-commerce solution user are recorded.

Settings for the workload:

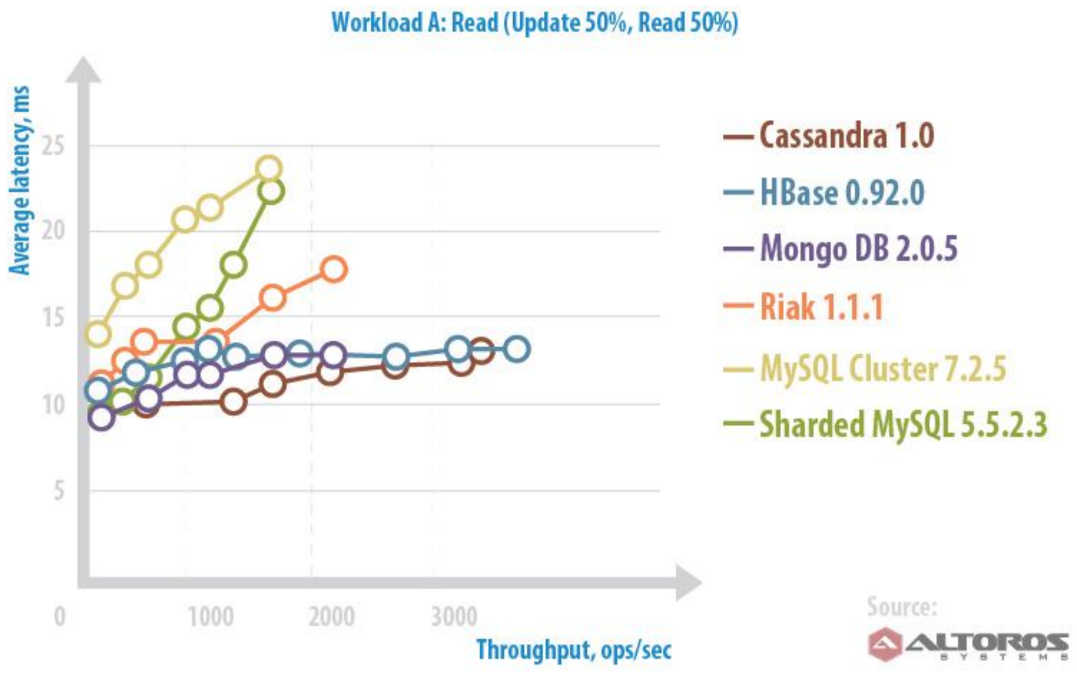
1) Read/update ratio: 50/50
2) Zipfian request distribution



During updates, HBase and Cassandra went far ahead from the main group with the average response latency time not exceeding two milliseconds. HBase was even faster. HBase client was configured with AutoFlush turned off. The updates aggregated in the client buffer and pending writes flushed asynchronously, as soon as the buffer became full. To accelerate updates processing on the server, the deferred log flush was enabled and WAL edits were kept in memory during the flush period.

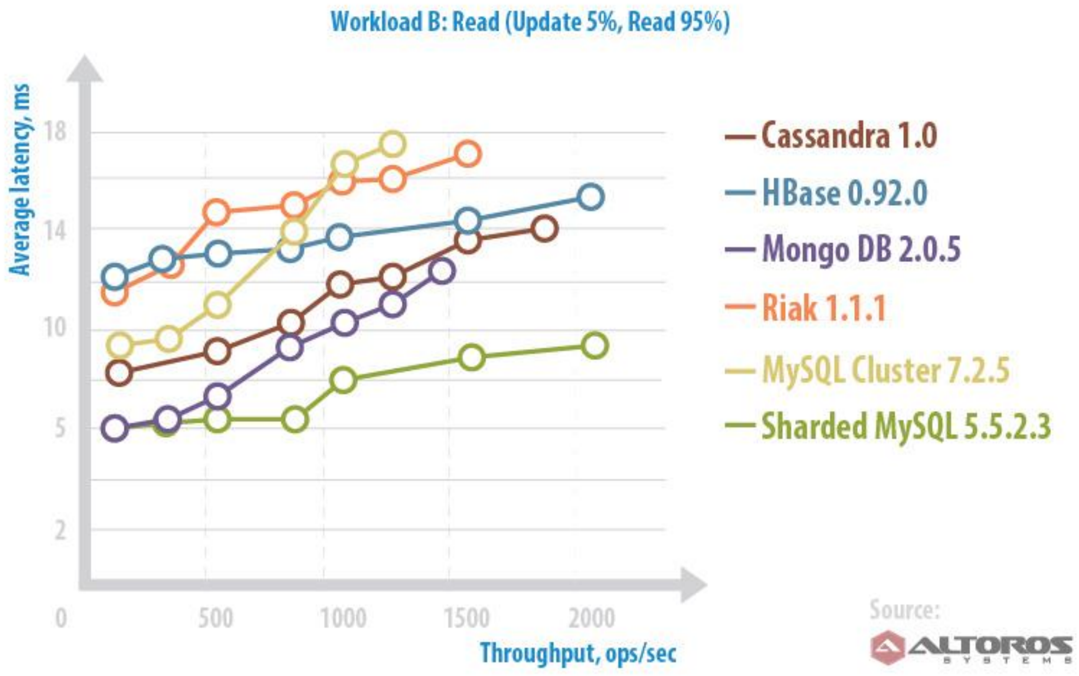
Cassandra wrote the mutation to the commit log for the transaction purposes and then an in-memory Memtable was updated. This is a slower, but safer scenario if compared to the HBase deferred log flushing.

*** Workload A: Read.** During reads, per-column family compression provides HBase and Cassandra with faster data access. HBase was configured with native LZO and Cassandra with Google's Snappy compression codecs. Although the computation ran longer, the compression reduces the number of bytes read from the disk.



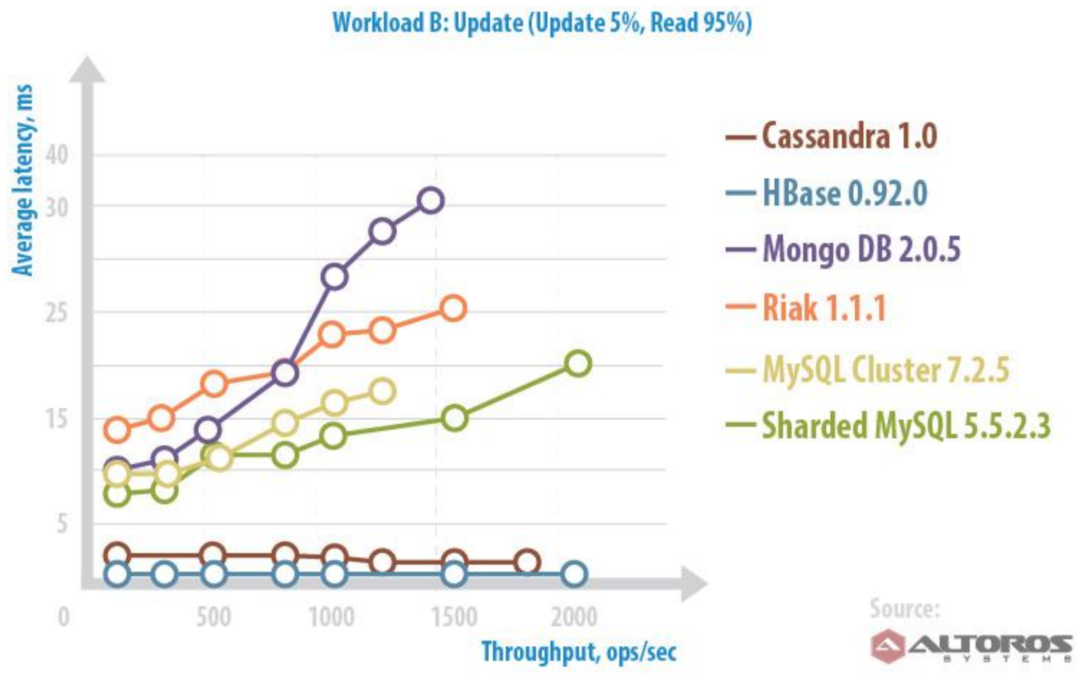
*** Workload B: Read-heavy mode.** Workload B consisted of 95% of reads and 5% of writes. Content tagging can serve as an example task matching this workload; adding a tag is an update, but most operations imply reading tags. Settings for the “read-mostly” workload:

1) Read/update ratio: 95/5
2) Zipfian request distribution



Sharded MySQL showed the best performance in reads. MongoDB—accelerated by the “memory mapped files” type of cache—was close to that result. Memory-mapped files were used for all disk I/O in MongoDB. Cassandra’s key and row caching enabled very fast access to frequently requested data. With the off-heap row caching feature added in version 0.8, it showed excellent read performance while using less per-row memory. The key cache held locations of keys in memory on a per-column family basis and defined the offset for the SSTable where the rows were stored. With a key cache, there was no need to look for the position of the row in the SSTable index file. Thanks to the row cache, we did not have to read rows from the SSTable data file. In other words, each key cache hit saved us one disk seek and each row cache hit saved two disk seeks. In HBase, random read performance was slower. However, Cassandra and HBase can provide faster data access with per-column-family compression.

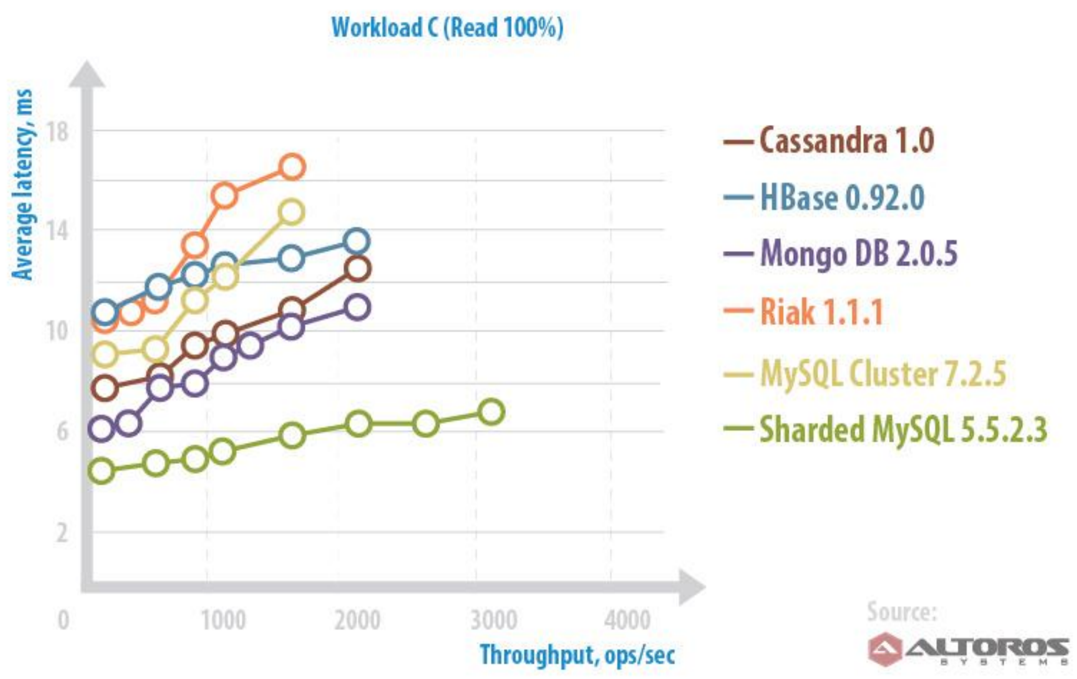
* **Workload B: Update.** Thanks to deferred log flushing, HBase showed very high throughput with extremely small latency under heavy writes. With deferred log flush turned on, the edits were first committed to the memstore. Then the aggregated edits were flushed to HLog asynchronously. On the client side, HBase write buffer cached writes with the autoFlush option set to true, which also improved performance greatly. For security purposes, HBase confirms every write after its write-ahead log reaches a particular number of in-memory HDFS replicas. HBase’s write latency with memory commit was roughly equal to the latency of data transmission over the network. Cassandra demonstrated great write throughput, since it first writes to the commit log—using the append method, which is a pretty fast operation—and then to a per-column-family memory store called Memtable.



* **Workload C: Read-only.** Settings for the workload:

1) Read/update ratio: 100/0
2) Zipfian request distribution

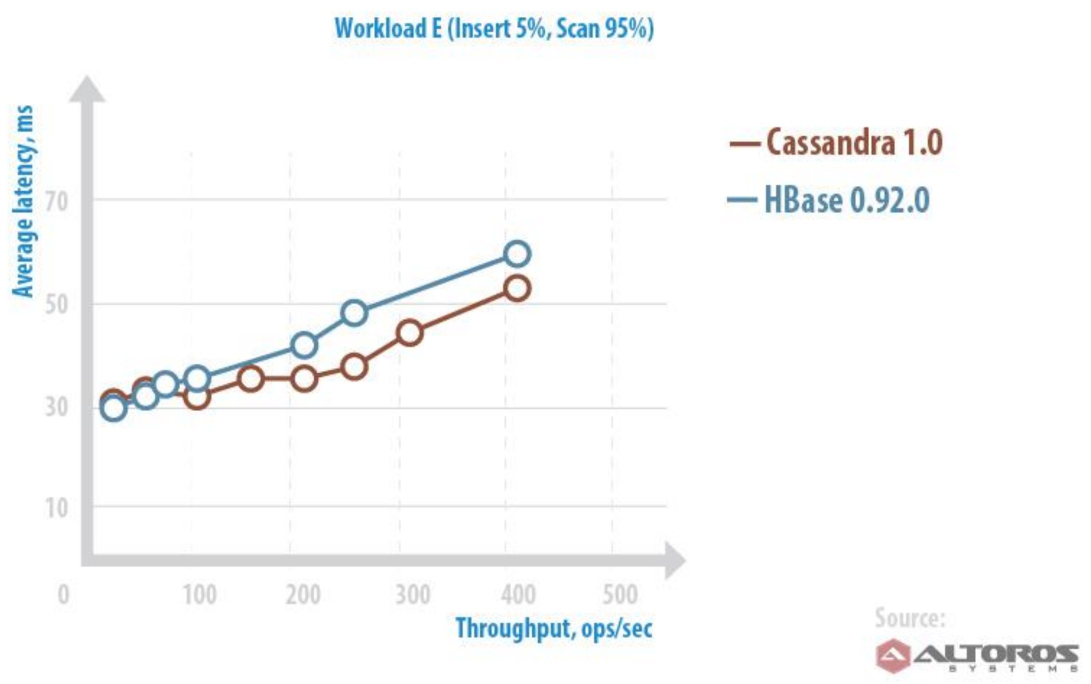
This read-only workload simulated a data caching system. The data was stored outside the system, while the application was only reading it. Thanks to B-tree indexes, sharded MySQL became the winner in this competition.



* **Workload E: Scanning short ranges.** Settings for the workload:

1) Read/update/insert ratio: 95/0/5
2) Latest request distribution
3) Max scan length: 100 records
4) Scan length distribution: uniform

HBase performed better than Cassandra in range scans. HBase scanning is a form of hierarchical fast merge-sort operation performed by HRegionScanner. It merges the results received from HStoreScanners (one per family), which, in their turn, merge the results received from HStoreFileScanners (one for each file in the family). If caching is turned on, the server will simply provide the number of specified records instead of bouncing back to the HRegionServer to process every record.



Cassandra's scan performance with a random partitioner has improved considerably compared to version 0.6, where this feature was initially introduced.

Cassandra's SSTable is a sorted strings table that can be described as a file of key-value string pairs sorted by keys and key-value pairs written in a particular order. To achieve maximum performance during range scans, we had to use an order-preserving partitioner. Scanning over ranges of order-preserving rows is super-fast. It is similar to moving a cursor through a continuous index. However, the database cannot distribute individual keys and corresponding rows over the cluster evenly, thus a random partitioner is used to ensure even data distribution. This is the default partitioning strategy in Cassandra. Random partitioning ensures good load balancing and provides some additional speed in range scans with an order preserving partitioner.

In MongoDB 2.5, the table scan triggered by the { "_id":{"$gte": startKey}} query showed a maximum throughput of 20 ops/sec with a latency of ≈ 1 sec.

The performance of MySQL Cluster was under 10 ops/sec with a latency of 400 ms. It is partitioned over the nodes in the cluster, so the system uses an optimizer to translate SQL commands into a query plan. The execution of this plan is divided among multiple nodes. For range scans, a B-tree index is used to make column comparisons in such expressions as >, <, or BETWEEN.

Sharded MySQL is based on key hashing on the connector side and does not support true range scans over a cluster. While a single shard did about 10 ops/sec, the whole sharded setup showed near 40 ops/sec with a latency of up to 400 ms. MyISAM caches index blocks but not data blocks. There can be an overhead due to re-reading data blocks from the OS buffer cache.

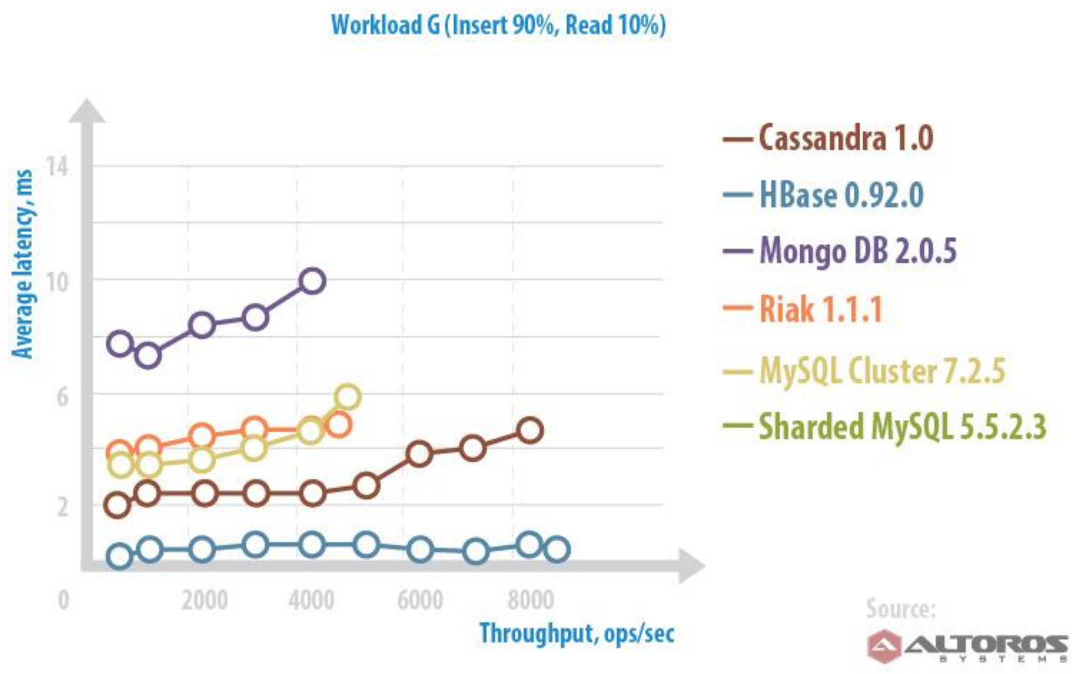
The Riak bitcask storage engine does not support range scans. This can be done through secondary indexes with eleveldb and special $key index referring to the primary key. Eleveldb showed insufficient performance that started to degrade after 50,000,000 records had been imported and we fell back to bitcask.

*** Workload G: Insert-mostly mode.** Settings for the workload:

1) Insert/Read: 90/10

2) Latest request distribution

HBase showed the best results under a workload that included large volumes of writes. Cassandra was second. The NDB engine of MySQL Cluster also managed intensive writes perfectly well.



5. Conclusion

As you can see, there is no perfect NoSQL database. Every database has its advantages and disadvantages that become more or less important depending on your preferences and the type of tasks.

For example, a database can demonstrate excellent performance, but once the amount of records exceeds a certain limit, the speed falls dramatically. It means that this particular solution can be good for moderate data loads and extremely fast computations, but it would not be suitable for jobs that require a lot of reads and writes. In addition, database performance also depends on the capacity of your hardware.

We hope this research will be useful to both developers working with NoSQL solutions and customers trying to choose a database. Altoros's R&D team will regularly revise and update information of this research to cover new databases and releases of the most popular products.

About the author:

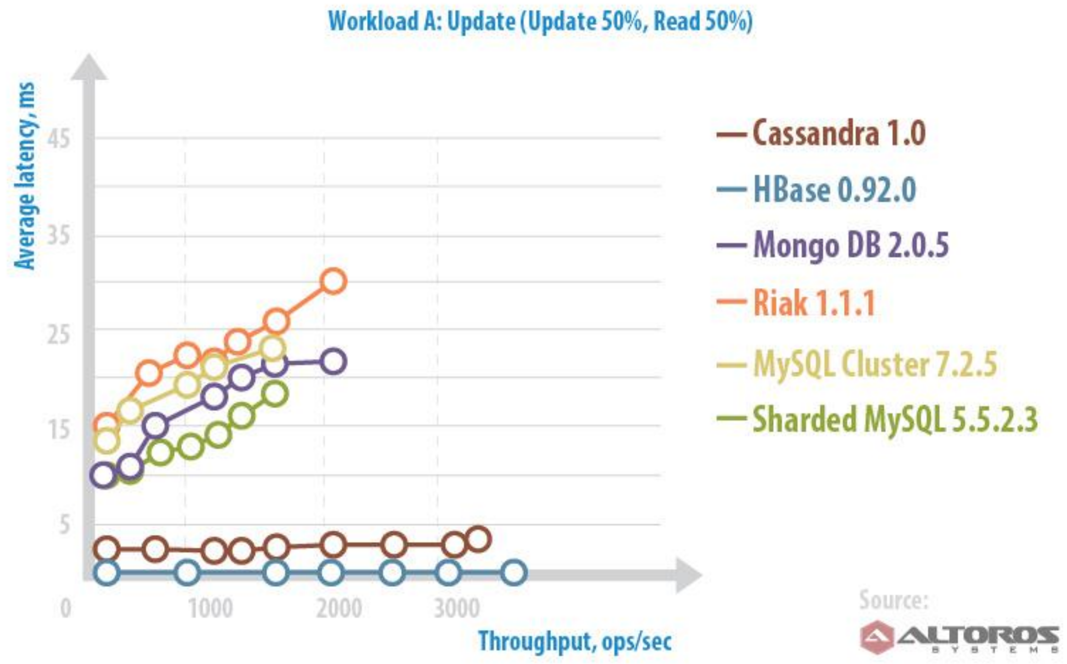
Sergey Bushik is a Senior R&D Engineer at Altoros. He has 7+ years of experience in implementation of Java-based projects that include big data processing, data mining, and Hadoop computations. Sergey has a number of certificates in Java and is a Sun Certified Enterprise Architect for the Java Platform. He is a regular speaker at international conferences—most recently, he delivered sessions at Big Data Meetup (Sunnyvale, CA), GOTO Copenhagen 2012, Hadoop Evening (Eastern Europe), etc.

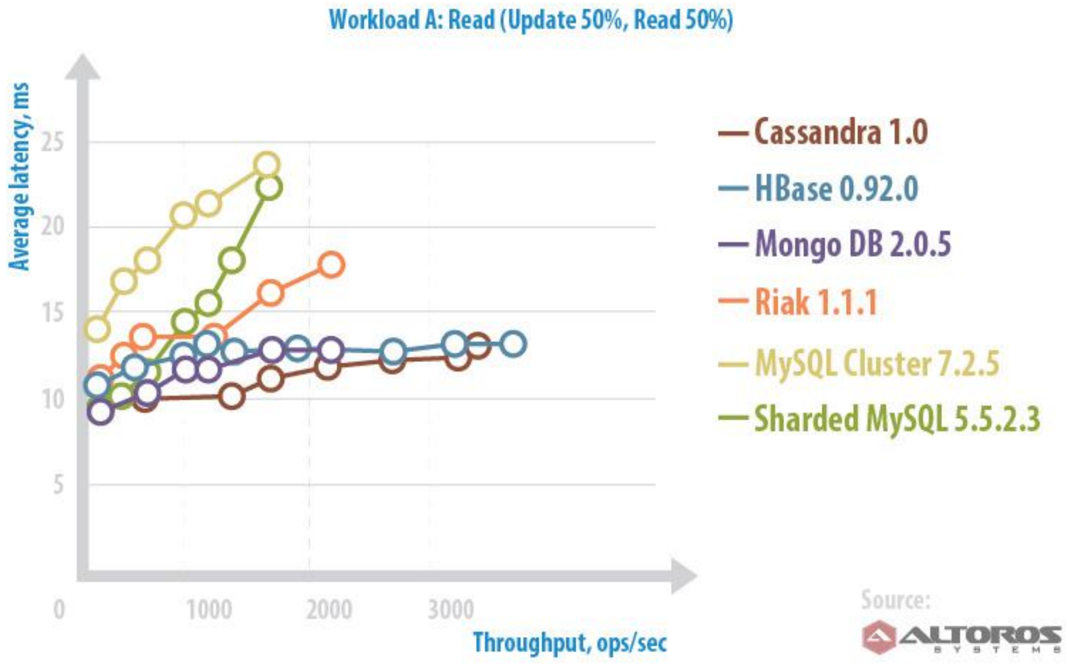
Liked this white paper?
Share it on the Web!

6. Appendix A

Workload A

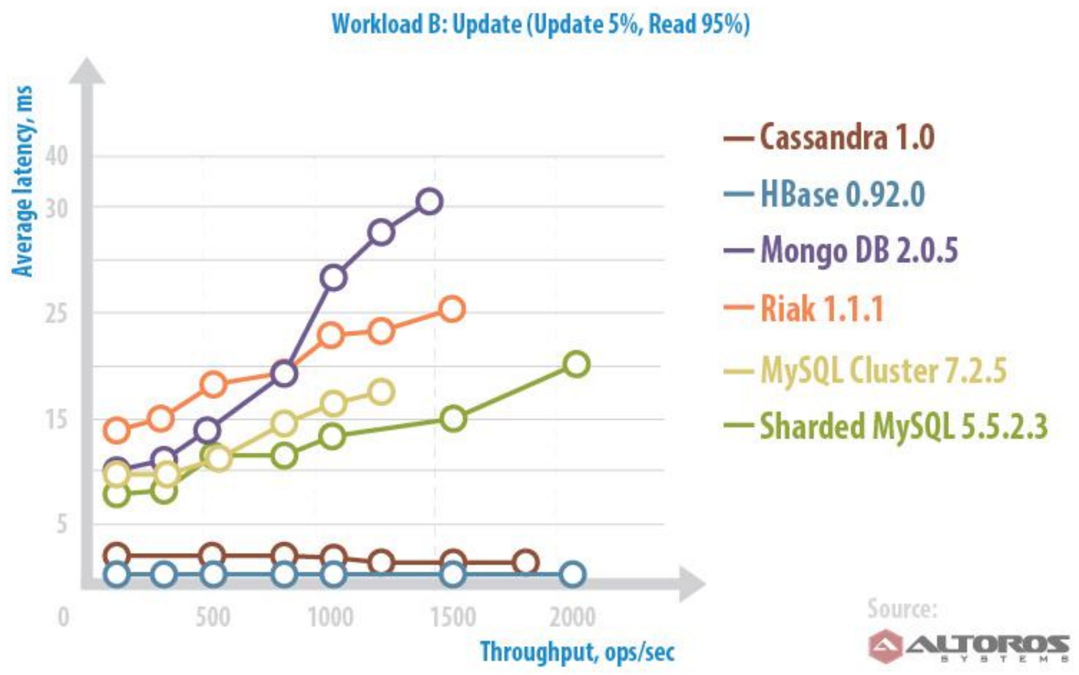
Workload A is an update-heavy workload, with a 50/50 read/update ratio. It simulates the database work, during which typical actions of an e-commerce solution user are recorded.

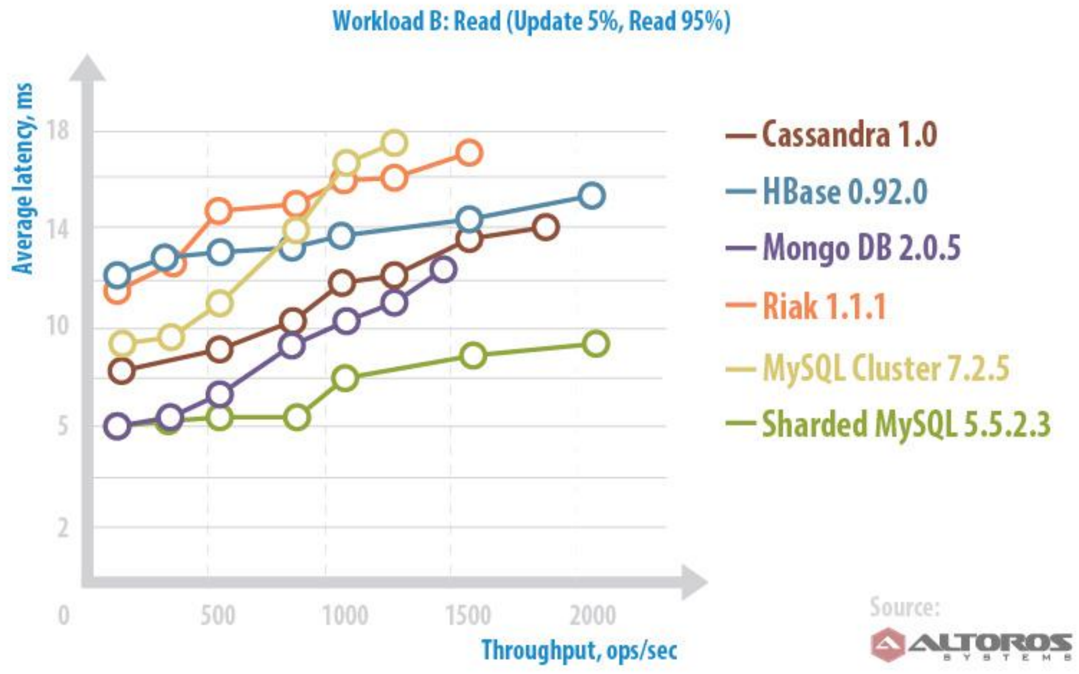




Workload B

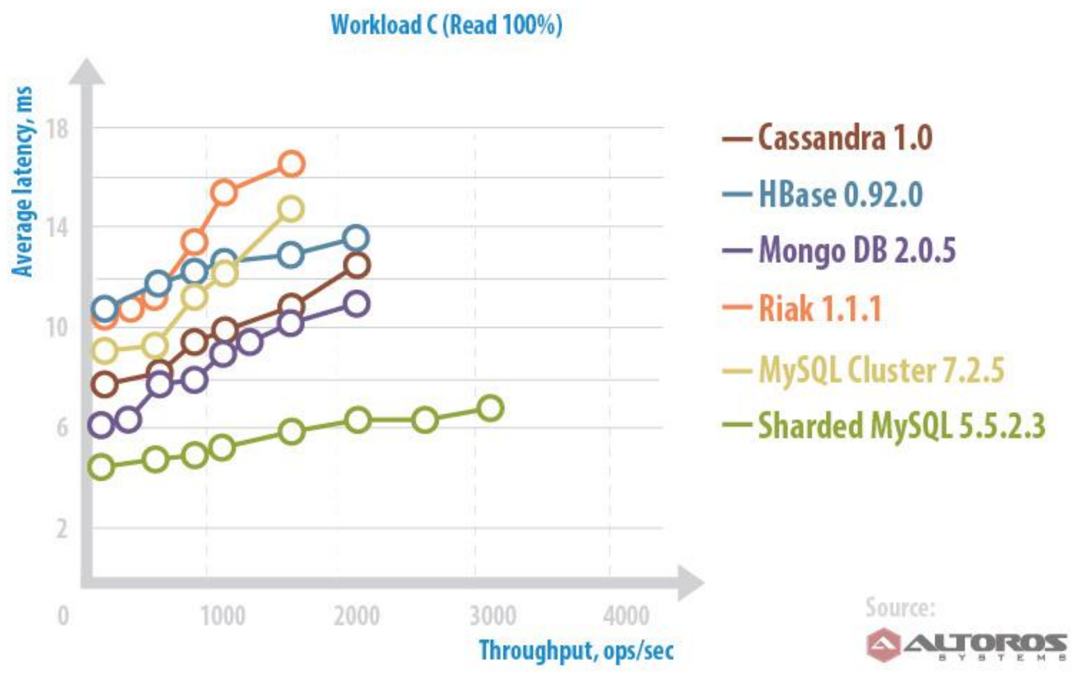
Workload B is a read-mostly workload that has 95/5 read/update ratio. It recaps content tagging, when adding a tag is an update, but most operations include reading tags.





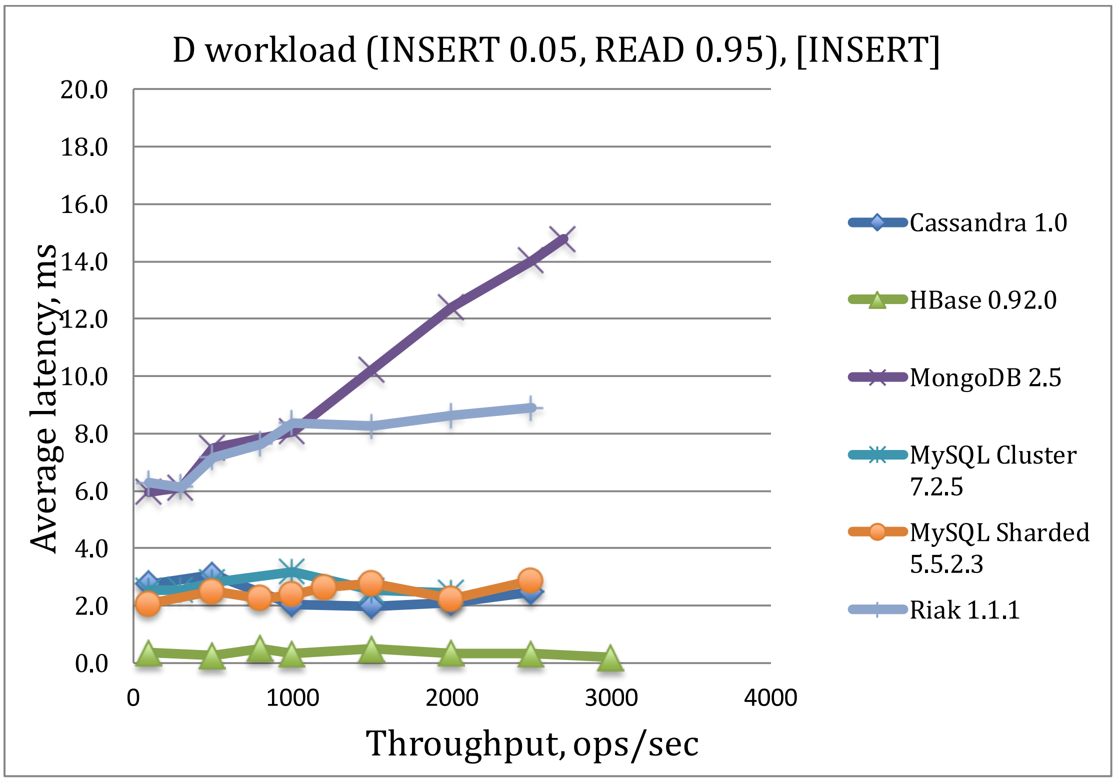
Workload C

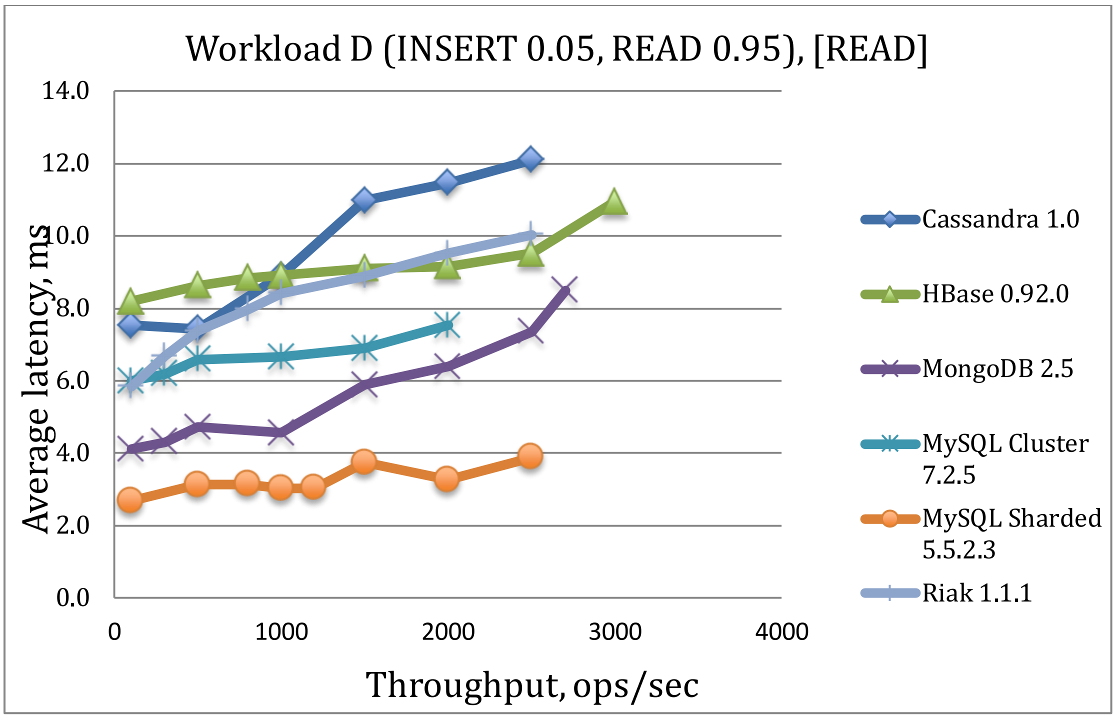
Workload C is a read-only workload that simulates a data caching layer, for example a user profile cache.



Workload D

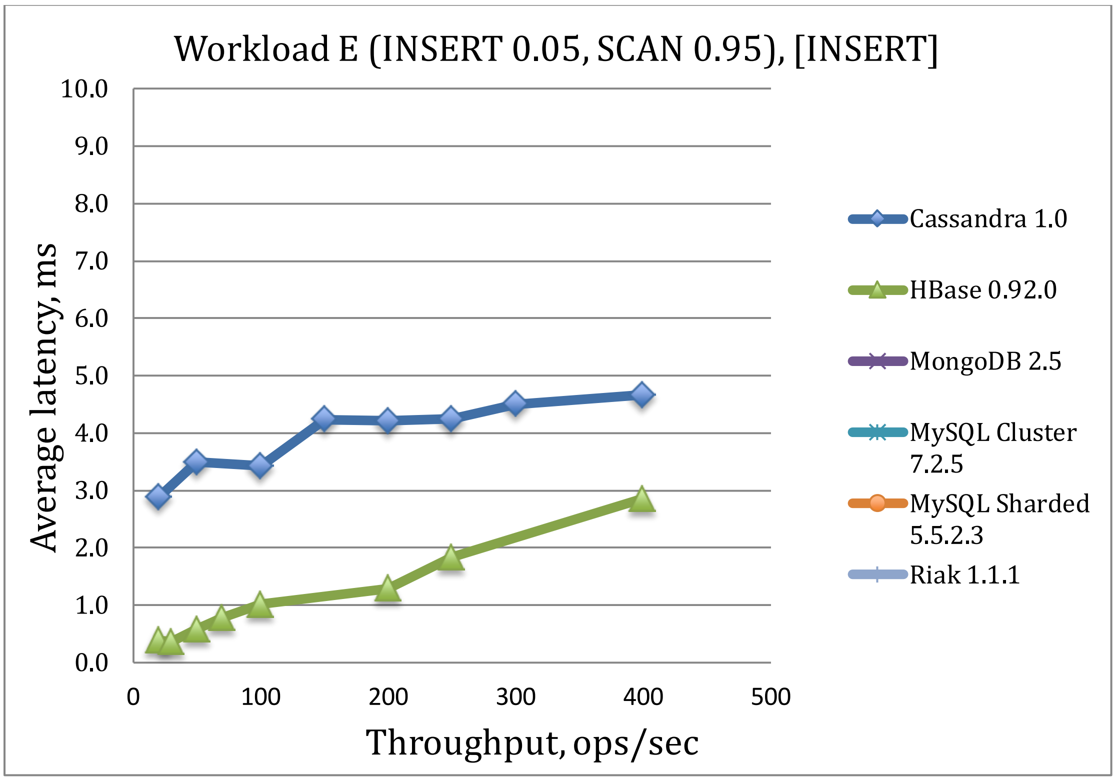
Workload D has 95/5 insert/read ratio. The workload simulates access to the latest data, such as user status updates or working with inbox messages first.

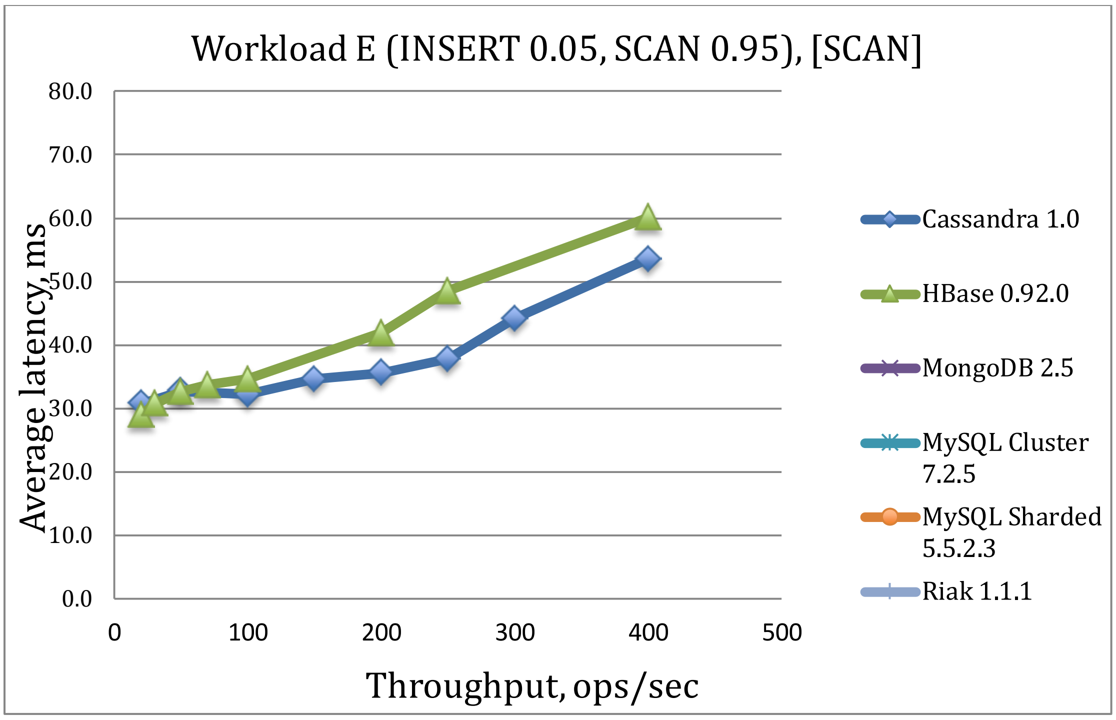




Workload E

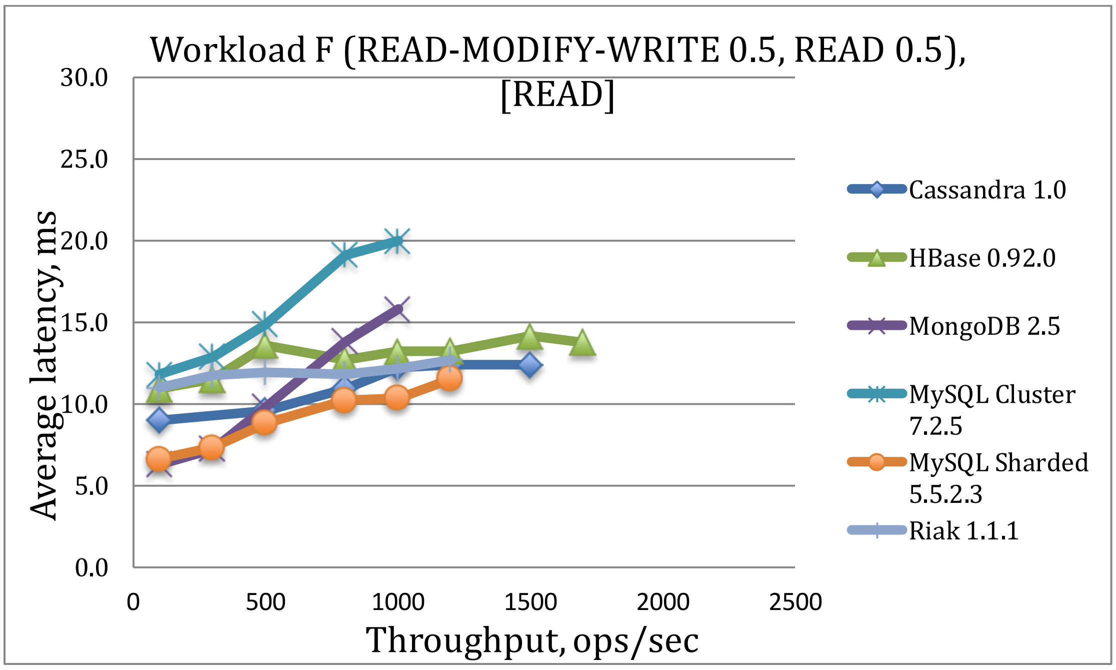
Workload E is a scan-short-ranges workload with a scan/insert percentile proportion of 95/5. It corresponds to threaded conversations that are clustered by a thread ID. Each scan is performed for the posts of a given thread.

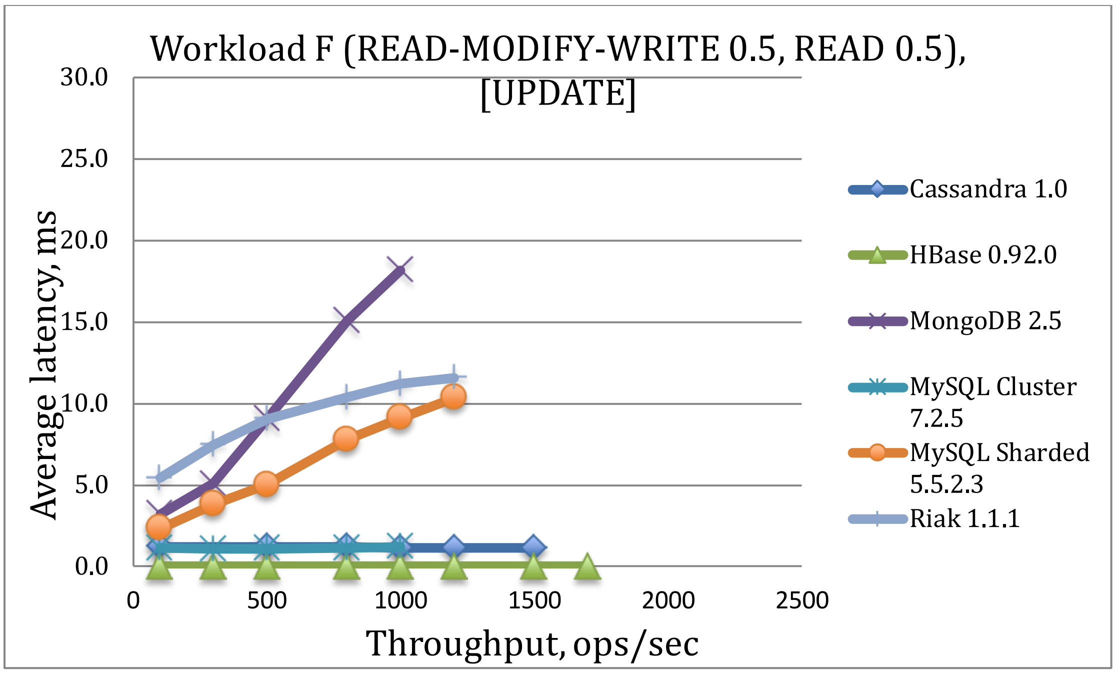


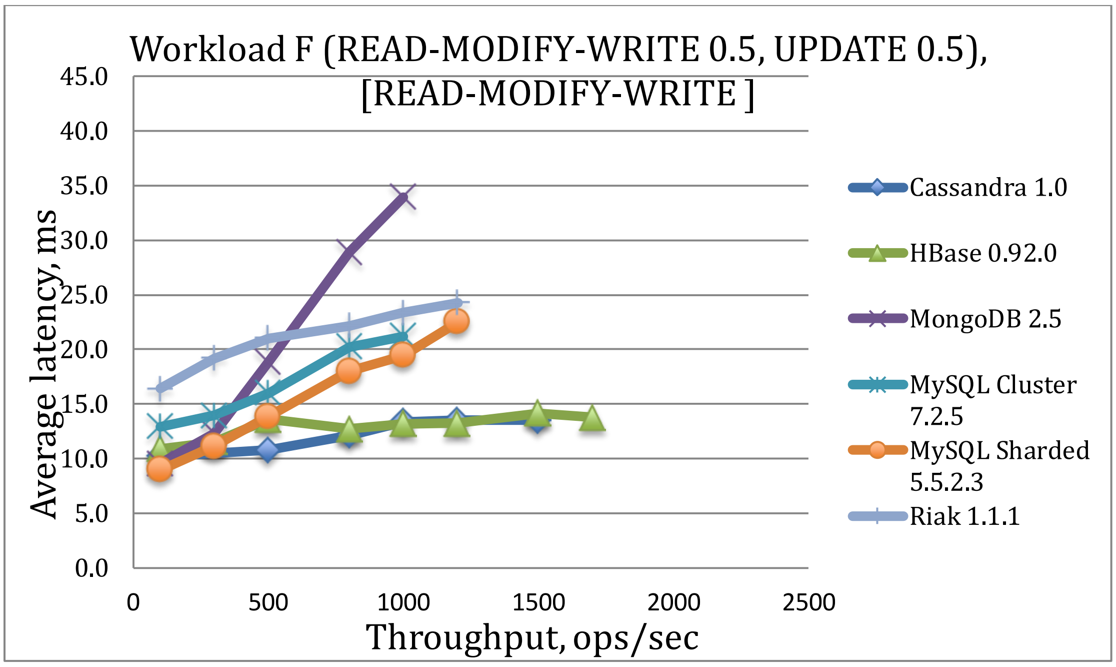


Workload F

Workload F has read-modify-write/read ops in a proportion of 50/50. It simulates access to user database, where user records are read and modified by the user. User activity is also recorded to this database.

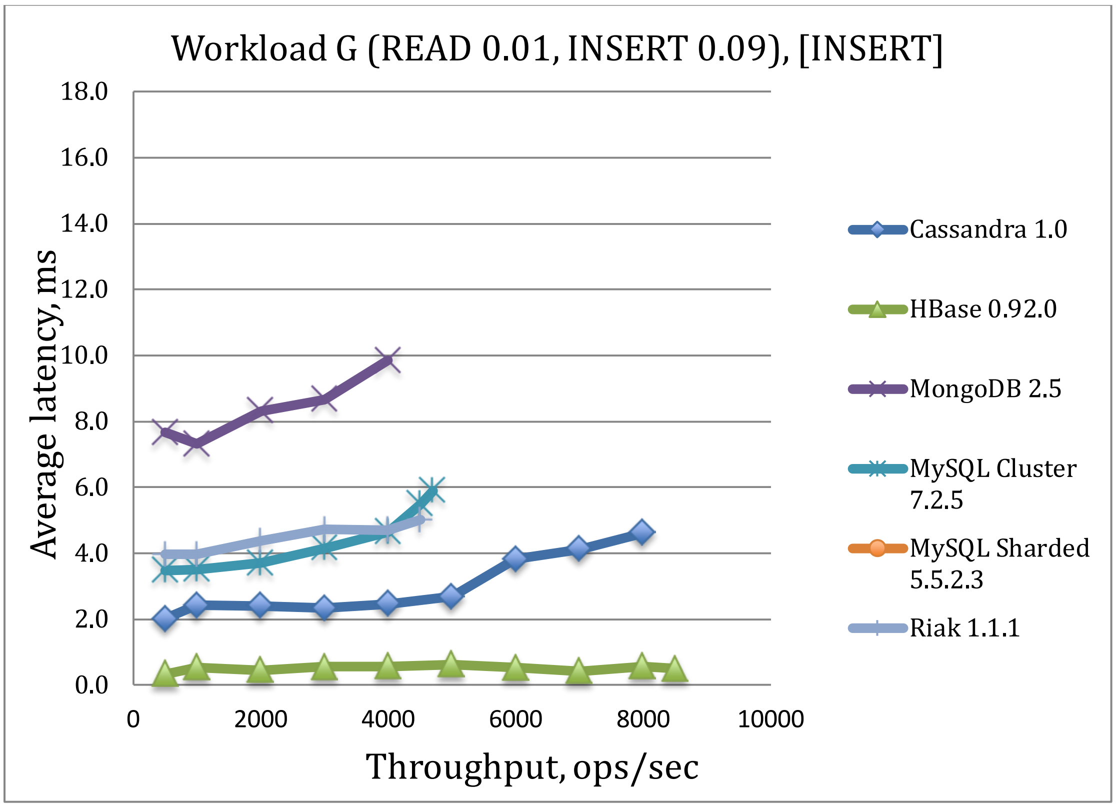


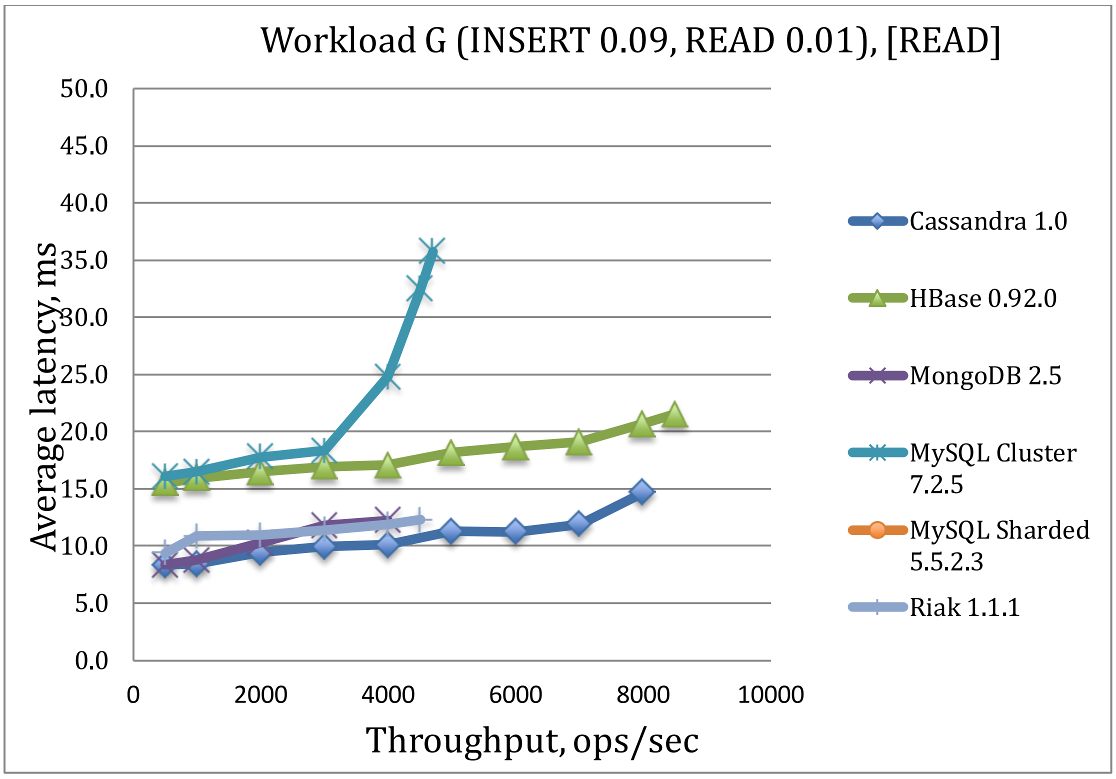

Workload F (READ-MODIFY-WRITE 0.5, READ 0.5), [UPDATE]
Average latency, ms
30.0
25.0
20.0
15.0
10.0
5.0
0.0
0
500
1000
1500
2000
2500
Throughput, ops/sec
Cassandra 1.0
HBase 0.92.0
MongoDB 2.5
MySQL Cluster 7.2.5
MySQL Sharded 5.5.2.3
Riak 1.1.1


Workload F (READ-MODIFY-WRITE 0.5, UPDATE 0.5), [READ-MODIFY-WRITE]
Average latency, ms
45.0
40.0
35.0
30.0
25.0
20.0
15.0
10.0
5.0
0.0
0
500
1000
1500
2000
2500
Throughput, ops/sec
Cassandra 1.0
HBase 0.92.0
MongoDB 2.5
MySQL Cluster 7.2.5
MySQL Sharded 5.5.2.3
Riak 1.1.1

Workload G

Workload G has a 10/90 read/insert ratio. It simulates data migration process or highly intensive data creation.





7. Appendix B

7.1 Cassandra

i. General Information

History	Cassandra was developed at Facebook to solve the issue with slow Inbox search across millions of messages. The database was open sourced in July 2008 featuring a schema inherited from Google's BigTable and running on an Amazon Dynamo-like infrastructure. In the following year the project moved to Apache Incubator. It was acknowledged as a top-level Apache product on February 17, 2010.
Notable Customers	Twitter, Digg, Reddit, Rackspace, Facebook
Best Use	Managing large streams of non-transactional data, such as Apache logs or application logs Applications that require a lot of writes–the database is consistent and provides fast response time under writes Real-time analytics & statistics
Type	Column-oriented
Language	Java
License	Apache License v 2.0
Commercial Support	DataStax, Impetus, Acunu, Riptapo, Cubet Technologies
Community Support	http://www.cassandrafaq.com/ http://www.topix.com/forum/city/cassandra-pa

ii. Technical Information

Language Bindings & Clients	Thrift is the official lowest-level API that provides access to Cassandra. This interface is primarily designed for client library developers. Thrift provides a software stack for building cross-language services and makes it possible to define data types and callable interfaces in a definition file. High-level clients already exist for:

	Python – Pycassa and Telephus;
	
	Java – Pelops, Hector, Kundera, Easy-Cassandra, Cassandrelle, and StorageProxy API intended for internal or highly specialized use-cases;
	
	Scala – Cascal;
	
	Clojure – clj-hector;
	
	.NET – Aquiles, cassandra-sharp, and FluentCassandra;
	
	Ruby – Fauna;
	
	PHP – Cassandra PHPH Client Library and phpcassa;
	
	Perl – Cassandra::Simple, Net::Cassandra, Net::Cassandra::Easy, and Cassandra::Lite;
	
	C++ – libQtCassandra.

Protocols	Thrift is a framework that combines a code generation engine and a software stack. Together they make it possible to develop scalable services with support for various programming languages. You can specify types of data in the definition file. Then it will be used as input data by the complier. The code generated with the help of the complier can be used to build RPC clients and servers required for effective cross-language communication.

iii. Operations

Querying	Thrift, also known as the high-level thrift wrapping client or CQL (Cassandra Query Language). For more details, refer to Appendix A. CQL
ACID, CAP	Cassandra does not fully support ACID semantics or Availability and Partitioning values of CAP. Trade-offs between consistency\latency are tunable in Cassandra.
Secondary Indexes	Cassandra 0.7 provides secondary indexes, which makes it possible to make queries by value. The queries can be processed automatically in the background and do not prevent reading or writing data. The key index, which is similar to a hash index, is the only type of index supported by Cassandra 0.7.0. It is planned to add support for bitmap indexes

	in the future.
MapReduce	MapReduce is available through custom input and output Hadoop formats.

iv. 4. Initializing a Cassandra Cluster on Amazon EC2.

EC2 Security Group for Cassandra

Step 1: In the Resources section of your Amazon EC2 Console dashboard, select Security Groups.

Step 2: Click Create Security Group. Enter a description and click Yes.

Step 3: Click Inbound and add rules for the ports.

Port	Rule Type	Description
22	SSH	Default SSH port
7000	Custom TCP Rule	Cassandra intra-node port
9160	Custom TCP Rule	Cassandra client port
7199	Custom TCP Rule	Cassandra JMX monitoring port

Deploy Amazon Image

In Amazon EC2 Console Dashboard, click Launch Instance. The Request Instances Wizard will be launched. Create an Amazon Instance with the Cassandra Security Group specified above.

Managing Amazon EC2 Disk

For information on how to enlarge an ext3 Amazon EBS volume, please, refer to Amazon EC2 Notes.doc

v. Installation Instructions

Prerequisites

In order to install Cassandra properly, it is preferable to use the most stable version of

Java 1.6. If the JVM developed by Sun Microsystems is used, u19 is the minimal requirement and u21 is recommended. Cassandra is compatible with IBM JRockit JVM. For CentOS Java can be installed with yum:

yum install java-1.6.0-openjdk.i386 (for 32-bit systems)

yum install java-1.6.0-openjdk.x86_64 (for 64-bit systems)

Download Distribution

Usually, there are several versions of the database available for download. We recommend using the latest stable version of the product. You can find all Cassandra releases available for download at **http://cassandra.apache.org/download**.

Download and extract the archive:

```
wget http://www.apache.org/dist/cassandra/1.0.7/apache-cassandra-1.0.7-bin.tar.gz
```

```
tar -zxvf apache-cassandra-1.0.7-bin.tar.gz
```

Move Cassandra to /usr/local and make a symbolic link:

```
$ sudo mv ~/apache-cassandra-1.0.7 /usr/lib
```

```
$ sudo ln -s /usr/lib/apache-cassandra-1.0.7 /usr/lib/apache-cassandra
```

Then create the directories required by Cassandra:

```
sudo mkdir -p /var/log/cassandra /var/lib/cassandra
```

```
sudo chown ec2-user /var/log/cassandra /var/lib/cassandra
```

Running a Single Node

Cassandra is designed to provide distributed storage on a cluster of nodes, but it can be also installed on a single machine. To load custom configuration, pass a VM parameter to Java when Cassandra starts up. To do this, assign a value to -Dcassandra.conf=<your value here>, i.e.

-Dcassandra.config=file:///home/ec2-user/cassandra.yaml.

Running a Cluster

Specifying Cluster Settings

Starting a Cassandra cluster is rather easy. You must specify such properties as seed nodes, initial token, and data partitioner for each node.

Cassandra configuration can be adjusted in the *conf/cassandra.yaml* file. It will connect to the Cluster Name specified in the *cluster_name:* property (the default name is "Test Cluster").

The Gossip protocol is used to exchange information between Cassandra nodes. Every new node should "know" at least one other seed node. Pick at least one or a few relatively stable nodes to be used as seeds.

Data can be allocated to certain nodes with the help of tokens. We used the default RandomPartitioner, which makes it possible to evenly distribute data across the nodes of the cluster. Prior to using a cluster for the first time, tokens should be assigned to every node in the cluster. This can be done by changing the *initial_token:* property.

For benchmarking, the tokens were generated with CassandraTokenGenerator, which is a utility class. For a cluster consisting of 4 nodes, the tokens will be as follows:

Node 0 token 0

Node 1 token 42535295865117307932921825928971026432

Node 2 token 85070591730234615865843651857942052864

Node 3 token 127605887595351923798765477786913079296

To add a new node to the cluster, in the configuration file, set the value of *auto_bootstrap:* to true. The node will join the ring and occupy the specified range. If the initial token is not explicitly defined in the configuration, the node will pick a token that will give it half of the keys from the node with the most disk space used.

For the first node cassandra.yaml will be as follows:

initial_token: value of the token or blank

rpc_address: rnd-nosql-cassandra1 or blank

listen_address: rnd-nosql-cassandra1 or blank

seeds: “rnd-nosql-node1”

For the rest of the nodes:

The same properties as on the first node plus

auto_bootstrap: true

Verify that the cluster is running as expected using the ring command of the node tool. This will display node status and show the ring:

```
# /usr/lib/apache-cassandra/bin/nodetool -host localhost ring
```

Creating a Data Schema

Cassandra is considered a schema-less data-store, but it requires some configuration work before benchmarking.

The Cassandra CLI client utility provides basic data definition functions and makes it possible to manage data within a Cassandra cluster. It is located in bin/cassandra-cli in binary installations.

```
# /usr/lib/apache-cassandra/bin/cassandra-cli -host localhost -port 9160
```

Step 1. Create a key space and name it “UserKeyspace”.

```
CREATE KEYSPACE UserKeyspace WITH placement_strategy = 'SimpleStrategy' AND
```

strategy_options = {replication_factor : 0} AND durable_writes = true;

Step 2. Create a column family and name it “UserColumnFamily”.

USE UserKeyspace;

Next, create a column family called “UserColumnFamily” without any columns. This structure fits for a dynamic family, since column names are not defined up front.

CREATE COLUMN FAMILY UserColumnFamily WITH comparator=UTF8Type AND key_validation_class=UTF8Type AND keys_cached=100000000 AND rows_cached=1000000 AND row_cache_provider='SerializingCacheProvider';

The column family was created with caching options for rows and keys enabled. Key cache is enabled by default only for 200,000 keys. Key cache helps to detect the position of the row on disk in the SSTable. If the row is accessed for the first time, the key cache would be initiated by the read operation. This ensures that further reads would be performed much faster. The key cache memory is limited to one disk seek per SSTable. If the row is entirely cached, the disk will not be accessed at all.

The compressor helps to fit large data sets into the memory and use disk space more efficiently, which results in improved read performance. Snappy compressor is a good option for processing read-heavy workloads, which provides a good balance between decompression speed and compression ratios.

The cluster is almost ready for benchmarking.

Memory Tuning

In the default configuration, half of all the available system memory in Cassandra is allocated to the JVM heap size.

This can be changed in cassandra.in.sh:

JVM_OPTS="-ea -XX:+UseConcMarkSweepGC -XX:+CMSIncrementalMode -Xmx6128m"

You can limit the JVM heap size using the following formula. Other settings will be automatically configured by the operating system’s file caching:

(memtable_total_space_in_mb) + (cache_size_estimate) + 1GB

memtable_total_space_in_mb property is a new feature that appeared in version 0.8. This setting must be configured for each node. The default value, which is equal to 1/3 of the maximum JVM heap size, will be used, if custom settings are not defined. If the value is set to zero, this disables the function and the system uses the old per CF thresholds. If you enable global settings and also define the per CF settings, both values will be used.

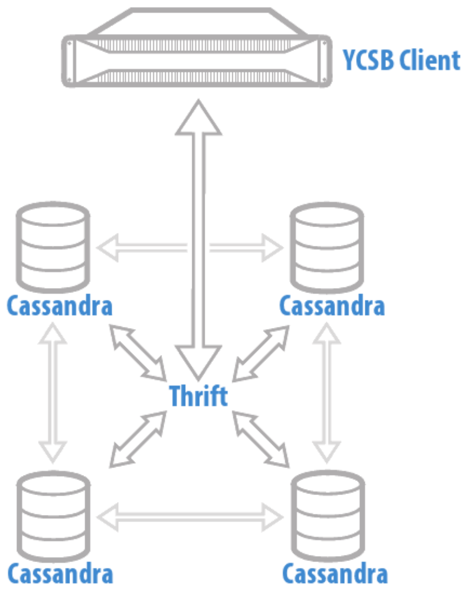
After version 1.0, you can store cached rows in native memory outside the Java heap. This minimizes the space required for per-row memory and also reduces the JVM heap size. As the result, the system shows better performance during garbage collection run by JVM. To use this new feature, the JNA library must be installed; otherwise, the old on-heap Cassandra cache provider will be used:

```
$ cd /usr/local/apache-cassandra/lib

$ curl
http://java.net/projects/jna/sources/svn/content/tags/3.3.0/jnalib/dist/platform.jar?rev=1208 > platform-3.3.0.jar

$ curl
http://java.net/projects/jna/sources/svn/content/tags/3.3.0/jnalib/dist/jna.jar?rev=1208 > jna-3.3.0.jar
```

Resulting Cluster



vi. CQL

ALTER COLUMNFAMILY – modifies a column family

ALTER COLUMNFAMILY <column family> ADD

<column> <validator>;

ALTER COLUMNFAMILY <column family> ALTER

<column> TYPE <validator>;

ALTER COLUMNFAMILY <column family> DROP

<column>;

BATCH – applies a single consistency level to many DML statements

BEGIN BATCH [USING <consistency> [AND

TIMESTAMP <timestamp>] [AND TTL <timetolive>]] <DML statements>;

APPLY BATCH;

CREATE COLUMN FAMILY – creates a new column family

CREATE COLUMNFAMILY <column family name>

(<key name> <type> PRIMARY KEY [, <name2>

<type2> [, <name3> <type3>, ...]]) [WITH

<option name> = <value> [AND <option name>

= <value> [...]];

CREATE INDEX – creates a new index on a column family

CREATE INDEX <index name> ON <column family> [(column name)];

CREATE KEYSPACE – creates a new keyspace

CREATE KEYSPACE <name> WITH strategy_

class = <strategy name> [AND strategy_

options:<option> = <value> [...]];

DELETE – deletes columns and/or rows from a column family

DELETE [<colname> [, <colname2> [, ...]]]

FROM <column family> [USING <consistency>

[AND TIMESTAMP <timestamp>]] WHERE <key

name> IN (<key1> [, <key2> [, ...]]);

DROP COLUMNFAMILY – drops/removes a column family

DROP COLUMNFAMILY <column family>;

Drops/removes a column family index

drop index <index name>;

DROP INDEX – drops/removes a column family index

DROP INDEX <index name>;

DROP KEYSPACE – drops/removes a keyspace

DROP KEYSPACE <keyspace>;

SELECT – retrieves data based on criteria

SELECT [FIRST <n>] [REVERSED] <select expr> FROM <column family> [USING <consistency>] [WHERE <clause>] [LIMIT <n>];

UPDATE – updates data in a column family

UPDATE <column family> [USING <consistency> [AND TIMESTAMP <timestamp>] [AND

TTL <timetolive>]] SET <column name 1> =

<value1> [, <column name 2> = <value2>

[, ...]] WHERE <key name> IN (<key1> [,

<key2> [, ...]]);

TRUNCATE – removes all data in a column family

TRUNCATE <column family>;

USE – switch to a keyspace to perform work

USE <keyspace name>;

Supported Consistency Levels

• CONSISTENCY ONE (default)

• CONSISTENCY QUORUM

• CONSISTENCY ALL

• CONSISTENCY EACH_QUORUM

• CONSISTENCY LOCAL_QUORUM

7.2 HBase

i. General Information

History	HBase started in 2006 after a publication by Google that described the data model of BigTable, a distributed storage system designed for scaling structured data of extra-large size. Initially, the project belonged to Powerset and was designed to process massive data during natural language search. Later, it attracted the interest of the company that owned Hadoop. Eventually, HBase became their sub-project in 2008. In 2010, Facebook implemented its new messaging infrastructure with HBase, which speeded-up the product's development and turned it into a top-level Apache project.
Notable Customers	Facebook, StumbleUpon, Yahoo
Best Use	Application log streams Real time analytics
Type	A column oriented, open-source, distributed database based on Google's BigTable. Built on top of HDFS, HBase makes it possible to look up and update records in large tables fast. HDFS is a distributed file system designed for storing large files. An HBase cluster can be expanded by adding servers, which are hosted on commodity class servers. HMaster is the name for the master server, which monitors all region server instances in a cluster and can be used as the interface for changing metadata. HRegionServer serves and manages regions. In a distributed cluster, a region server runs on a data node.
Language	Java
License	Apache License v 2.0
Commercial Support	Cloudera
Community Support	http://www.cloudera.com/blog/category/hbase/

ii. Technical Information

Language Bindings	Java can be accessed directly through Java API.

& Clients	You can work with non-Java clients using Thrift API & Stargate Rest servers that come with HBase.
Protocols	Stargate REST server works as a daemon, which starts an embedded Jetty servlet container. Thrift API is cross-platform which means it is universal. In addition, it is more lightweight than REST, so it works faster.

iii. Operations

Querying	HBase supports secondary indexes and provides filter queries for full scanning of the tables. Both ways are available through either a Java based client or the Thrift API & Stargate REST interface.
ACID, CAP	HBase values partition tolerance and consistency of the CAP theorem. It also guarantees certain specific ACID properties. **Atomicity:** All mutations within a row are kept atomic. A row mutation that returns the “success” code has been completed successfully and an operation with a "failure" code has failed. **Consistency:** Any access to API returns all rows, i.e. all complete rows that have ever existed in the table. A scan neither provides a consistent view of a table, nor does it reveal snapshot isolation. **Isolation:** Resembles a “read committed” isolation level in relational databases (get and scan will return the latest persisted data for the moment the records were fetched) **Durability:** All visible data is durable data, meaning that read will never return data that has not been made durable on disk. All data that is visible in HBase is durable. This is guaranteed by the Atomicity principle described above. Data that is not durable on disk will never be displayed as the result of a read operation.
Secondary Indexes	Indexing by column values

MapReduce	Available through Hadoop. Hadoop makes it possible to input data into HBase and save it in a format compatible with MapReduce.

Initializing an HBase Cluster on Amazon EC2.

EC2 Security Group for HBase

Step 1: In My Resources section of your Amazon EC2 Console, select Security Groups.

Step 2: Click Create Security Group. Fill out the description and click Yes.

Step 3: Click Inbound and add rules for the following ports.

Port	Rule Type	Description
22	SSH	Default SSH port
	HBase	
60000	Custom TCP Rule	The port master should bind to (hbase.master.port)
60010	Custom TCP Rule	HTTP port for the master Web UI. You shouldn't add it if you don't want the info server to be exposed. (hbase.master.info.port)
60020	Custom TCP Rule	Port an HBase RegionServer binds to (hbase.regionserver.port)
60030	Custom TCP Rule	HTTP port for the HBase RegionServer Web UI (hbase.regionserver.info.port)
	ZooKeeper	
2888	Custom TCP Rule	Port used by ZooKeeper peers to connect to other peers (hbase.zookeeper.peerport)
3888	Custom TCP Rule	Port used by ZooKeeper for the leader election (hbase.zookeeper.leaderport)
2181	Custom TCP Rule	Port for connecting clients (hbase.zookeeper.property.clientPort)
	HDFS	
8020	Custom TCP Rule	File system metadata operations (fs.default.name)
50010	Custom TCP Rule	DataNode transfer (dfs.datanode.address)
50020	Custom TCP	Block metadata operations and recovery

	Rule	(dfs.datanode.ipc.address)
50100	Custom TCP Rule	Backup node file system metadata operations (dfs.backup.address)
50070	Custom TCP Rule	NameNode HTTP port (dfs.http.address property)
50075	Custom TCP Rule	DataNode HTTP port (dfs.datanode.http.address)
50090	Custom TCP Rule	Secondary NameNode HTTP port (dfs.secondary.http.address)
	Map/Reduce	
8020	Custom TCP Rule	Job submission, task tracker heartbeats port, which is the part of URL hdfs://hostname:8020/ (mapred.job.tracker)
	Custom TCP Rule	TaskTracker port communicating with child jobs (mapred.task.tracker.report.address)
50030	Custom TCP Rule	JobTracker HTTP port (mapred.job.tracker.http.address)
50060	Custom TCP Rule	TaskTracker HTTP port (mapred.task.tracker.http.address)

Installation Instructions

Installing Java

HBase requires JDK version 1.6 or higher to be installed. Download an appropriate package from Java SE Downloads and run the binary:

```
$ wget http://download.oracle.com/otn-pub/java/jdk/6u31-b04/jdk-6u31-linux-x64.bin

$ chmod a=rwx ./jdk-6u31-linux-x64.bin

$ ./jdk-6u31-linux-x64.bin

$ sudo mv ./jdk1.6.0_31 /usr/lib/jvm/jdk1.6.0_31
```

Add JAVA_HOME environmental property to ~/.bash_profile, refresh bash environment (relogin current user), and configure alternatives:

```
$ sudo nano ~/.bash_profile

# JAVA_HOME=/usr/lib/jvm/jdk1.6.0_31

# export JAVA_JOME
```

```
$ sudo alternatives --install /usr/bin/java java $JAVA_HOME/bin/java 2

$ sudo alternatives --config java
```

Installing HBase

We use a Cloudera HBase distribution. It simplifies installation of both Hadoop and HBase. Before installing HBase, you must download the CDH3 repository. Detailed instructions for various platforms are available here. To install the package locally on Amazon Linux, which is a custom CentOS 6 distribution, do the following:

```
$ sudo yum --nogpgcheck localinstall
http://archive.cloudera.com/redhat/6/x86_64/cdh/cdh3-repository-1.0-1.noarch.rpm
```

To install a new version of HBase on a system with yum package manager, do the following:

```
$ sudo yum install hadoop-hbase

$ rpm -ql hadoop-hbase
```

Installation scripts for different systems are available on this page.

Make the hbase/logs directory writable for the current user:

```
$ cd /usr/lib/hbase

$ sudo chmod a=rwx logs pids
```

Configuration Steps.

Single-node Mode

In the default configuration HBase is adjusted for standalone mode. The local HBase file system can be used to run a single-node standalone instance. You should edit conf/hbase-site.xml and define the directory that HBase will add data to. It can be done by editing the “hbase.rootdir” property:

```
<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

  <property>

    <name>hbase.rootdir</name>
    <value>file:///tmp/hbase-${user.home}/hbase</value>
  </property>
</configuration>
```

The default value of “hbase.rootdir” is file:///tmp/hbase-${user.home}/hbase.

Run the HBase server with the following script:

```
$ cd /usr/lib/hbase/
$ bin/start-hbase.sh
```

Open the HBase shell and enter 'help<RETURN>' to get the list of supported commands:

```
$ bin/hbase shell
```

Distributed Mode

Make sure that you can connect to a localhost via the SSH protocol without entering a password. After that, run the following commands:

```
$ ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
$ cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
```

Also, add id_dsa.pub to the authorized keys for each slave node.

Template configuration files for Hadoop and HBase in the project sources:

/hbase-0.92.0/hadoop-conf/

/hbase-0.92.0/hbase-conf/

Override the default configuration with these files and specify the following:

- Name node host and port (fs.default.name)
- Job tracker host and port (mapred.job.tracker)
- Slaves file that contains the names of all the hosts in the cluster

Create a directory for NameNode data as specified by "dfs.name.dir" in hdfs-site.xml (if not using default):

```
$ sudo mkdir -p /var/lib/hdfs/name

$ sudo chmod 700 /var/lib/hdfs/name

$ sudo chown ec2-user /var/lib/hdfs/name
```

Add the required access permissions to logs, pid directories. Update the /etc/hosts file with appropriate mapping of slave hosts and run the NameNode service:

```
$ cd /usr/lib/hadoop-0.20

$ sudo chmod a=rwx logs pids
```

Format NameNode using the following command (you will get an upper-case letter Y as the answer):

```
$ bin/hadoop namenode -format
```

Then start the service:

```
$ bin/hadoop-daemon.sh start namenode
```

The /hbase directory should be created in HDFS prior to launching the HBase Master. HBase master does not possess the necessary rights and thus cannot create top level directories.

```
$ bin/hadoop fs -mkdir /hbase

$ bin/hadoop fs -chown hbase /hbase
```

On each data node, create a directory for data blocks and add the required access permissions for the directory (if not using default):

```
$ sudo mkdir -p /var/lib/hdfs/data

$ sudo chmod 700 /var/lib/hdfs/data

$ sudo chown ec2-user /var/lib/hdfs/data
```

Remember to update the /etc/hosts file with master and slave node mappings, update the configuration files for Hadoop, and run datanode:

```
$ cd /usr/lib/hadoop-0.20

$ sudo chmod a=rwx logs pids

$ bin/hadoop-daemon.sh start datanode
```

Add a new node to the conf/slaves list of the master node and execute:

```
$ bin/hadoop dfsadmin -refreshNodes
```

Creating the Database Schema

Open HBase shell and create a schema to be used with the benchmarking connector:

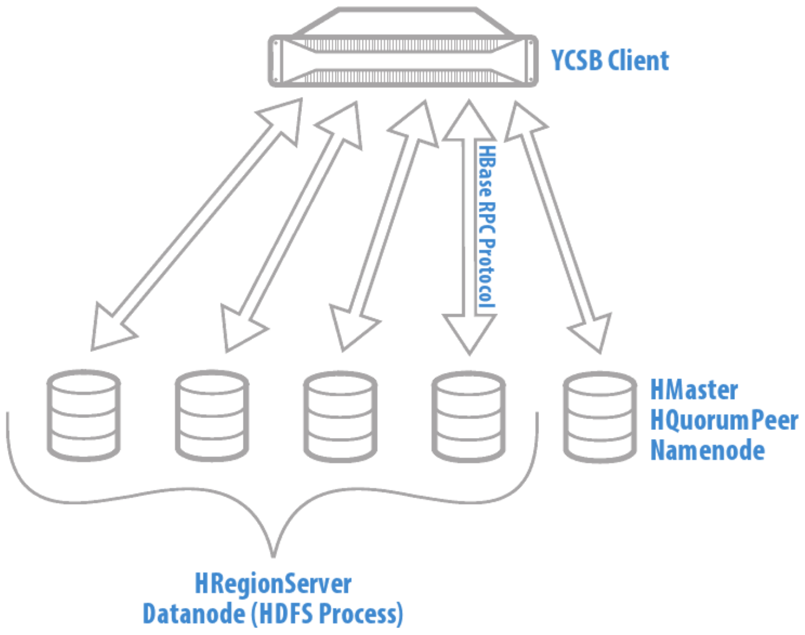
```
create 't1', {NAME => 'f1', BLOOMFILTER => 'ROW', REPLICATION_SCOPE => '0',
COMPRESSION => 'LZO', VERSIONS => '3', TTL => '2147483647', BLOCKSIZE =>
'16384', IN_MEMORY => 'true', BLOCKCACHE => 'true', DEFERRED_LOG_FLUSH =>
'true'}

disable 't1'

alter 't1', METHOD => 'table_att', MAX_FILESIZE => '1073741824'

enable 't1'
```

Resulting Cluster



Developer Notes

Open Files Limit

HBase works with multiple files simultaneously. This is why it might be inconvenient to use it with *nix systems, where you can only have 1024 files open at a time. When the system has to manage a heavy load, the error message "Too many open files" appears. Refer to this section to modify the limit in HBase book.

For Amazon Linux edit /etc/security/limits.conf and add soft/hard limits for nproc:

```
ec2-user      soft   nproc       8192
ec2-user      soft   nofile     65536
ec2-user      hard   nproc       8192
ec2-user      hard   nofile     65536
```

Durable Sync

HBase requires HDFS with a durable sync implementation to keep data. You should enable sync explicitly by changing dfs.support.append to true in hbase-site.xml on both the client and server side:

```
<property>
   <name>dfs.support.append</name>
   <value>true</value>
</property>
```

You can find more information about support for this feature in different versions of HBase in documentation provided with Hadoop.

dfs.datanode.max.xcievers

In a Hadoop HDFS data node, only a limited number of files can be served at a time:

```
<property>
   <name>dfs.datanode.max.xcievers</name>
   <value>4096</value>
</property>
```

Column Family Name

In HBase, row, column, and version are used to specify a cell. A column name consists of its column family name prefix and a qualifier. Since column family name is used with each cell as part of its address, column family name should be shortened as much as possible, preferably to one character.

Puts and Auto Flush

When auto flush of HTable is disabled, puts are sent to the write buffer. If it’s already filled, they are passed to the region servers. For high volume puts (such as loading database) you might consider disabling auto flush and making the write buffer larger, which is only 2MB by default. Commits should be flushed before an HTable instance is discarded to ensure that no data will be lost.

Compression

With compression enabled, the store HFile applies a compression algorithm to blocks as they are created during flushes and compactions. This means that the blocks should be decompressed for reading.

Despite the fact that it requires extra time to read the records, compression provides a number of advantages that justify its use:

• Compression decreases the size of the data blocks that are written to or read from HDFS.

• Compression enhances network bandwidth and helps to manage disk space more efficiently.

• Compression reduces the size of data to be read when issuing a read.

Out of the box, HBase only provides Gzip compression, which is rather slow. In order to improve performance significantly, a real-time compression library, such as lossless data compression algorithm LZO, should be used. LZO is distributed under the GPL. To get it working, you must build lzo2 from source in the 64-bit mode:

```
$ wget http://www.oberhumer.com/opensource/lzo/download/lzo-2.06.tar.gz

$ tar zxf lzo-2.06.tar.gz

$ rm lzo-2.06.tar.gz

$ cd lzo-2.06

$ sudo yum install gcc

$ CFLAGS="-fno-strict-aliasing -fPIC" ./configure

$ make

$ sudo make install

$ cd ..

$ sudo rm -rf lzo-2.06/

$ sudo yum install lzo.x86_64
```

Check out the native connector library:

```
$ sudo yum install svn

$ svn checkout http://svn.codespot.com/a/apache-extras.org/hadoop-gpl-compression/trunk/ hadoop-gpl-compression-read-only

$ sudo yum install ant-nodeps

$ sudo mkdir /usr/lib/jvm-exports/jdk1.6.0_31

$ sudo cp /usr/lib/jvm-exports/java/* /usr/lib/jvm-exports/jdk1.6.0_31/

$ cd hadoop-gpl-compression-read-only/
```

Build the native connector library. To perform compilation for a 64-bit machine on Linux with the gcc compiler, do the following:

```
$ CFLAGS="-m64"

$ ant compile-native

$ ant jar

$ sudo cp build/hadoop-gpl-compression-0.2.0-dev.jar /usr/lib/hbase/lib/
$ sudo cp build/native/Linux-amd64-64/lib/libgplcompression.*
/usr/lib/hbase/lib/native/Linux-amd64-64/

$ cd ..

$ rm -rf ./hadoop-gpl-compression-read-only/
```

To test, if compression is properly enabled, run:

```
$ touch lzo.txt

$ /usr/lib/hbase/bin/hbase org.apache.hadoop.hbase.util.CompressionTest `pwd`/lzo.txt
lzo

$ rm -f lzo.txt
```

7.3 MongoDB

General Information

History	Development of MongoDB began in fall of 2007, when 10gen's team started working on a cloud platform as a service solution with functionality similar to Google's App Engine. The decision to open source the product and turn it into a standalone system increased the popularity of the database and facilitated its rapid spread. Now, MongoDB is generally distributed under the AGPL license. A commercial license can be obtained from 10gen. Starting from version 1.4 launched in March 2011, the product is recognized as ready for production use.
Notable Customers	Disney, SAP, Forbes, GitHub
Best Use	Archiving and event logging Content management systems Gaming
Type	Document oriented
Language	C++
License	MongoDB is licensed under the AGPL v3.0. Commercial licenses are also available from 10gen. Most drivers are licensed under the Apache License v2.0.
Commercial Support	10gen
Community Support	http://blog.mongodb.org

	http://twitter.com/mongodb http://groups.google.com/group/mongodb-user?pli=1

Technical Information

Language Bindings & Clients	MongoDB uses Mongo Wire Protocol to communicate with the database server. BSON is used to transfer data to and from clients. Drivers are available for application developers using different programming language platforms: C, C++, Erlang, Haskell, Java, JavaScript, .NET (C#, F#, PowerShell, etc), Perl, PHP, Python, Ruby, Scala.
Protocols	Mongo Wire Protocol is based on the request-response principle and clients interact with the database through a TCP/IP socket.

Operations

Querying	Mongo Query language
ACID, CAP	ACID semantics are not supported. This database values consistency and partition tolerance of the CAP theorem.
Secondary Indexes	Indexing of embedded element, compound key
Map/Reduce	Available

Initializing a MongoDB Cluster on Amazon EC2

EC2 Security Group for MongoDB

Step 1: In My Resources section of your Amazon EC2 Console Dashboard, select Security Groups.

Step 2: Click Create Security Group. Fill out a description and click Yes.

Step 3: Click Inbound and add rules for the following ports.

Port	Rule Type	Description
22	SSH	Default SSH port
27017	Custom TCP Rule	Standalone mongod or mongos
27018	Custom TCP Rule	Shard server (mongod --shardsvr)

27019	Custom TCP Rule	Config server (mongod --configsvr)
28018	Custom TCP Rule	Web stats page for mongod

Installation

MongoDB sharding components include individual shards (mongod's), config servers, and mongos processes. Four shard nodes, an additional node for the configuration server, and a mongos router were used for benchmarking.

Name	Description
rnd-nosql-client	Yahoo Cloud Serving Benchmark client
rnd-nosql-server-router	A mongos process combines both routing and coordination processes. As a result, various components of the cluster work as a unified system.
rnd-nosql-server1	Core mongod database process 1
rnd-nosql-server2	Core mongod database process 2
rnd-nosql-server3	Core mongod database process 3
rnd-nosql-server4	Core mongod database process 4

Download the required version of MongoDB and extract the archive. We used version 2.0.5:

```
$ curl http://downloads.mongodb.org/linux/mongodb-linux-x86_64-2.0.5.tgz >
~/mongodb-linux-x86_64-2.0.5.tgz

$ tar xzf ~/mongodb-linux-x86_64-2.0.5.tgz

$ sudo mv ~/mongodb-linux-x86_64-2.0.5/ /usr/lib/mongodb-2.0.5
```

For a basic database configuration, create a directory for the database and log files:

```
# /var/lib/mongodb/ is used by default for the database and /var/log/mongodb for logs as
specified in mongo-bin/set-mongo-env.sh

$ sudo mkdir /var/lib/mongodb/ /var/log/mongodb/
```

Make sure that mongo process has appropriate permissions to access these directories:

```
$ sudo chown ec2-user /var/lib/mongodb/ /var/log/mongodb/
```

If you have to mount the file system, please check that noatime and nodiratime settings are enabled:

```
echo "/dev/md0 /raid xfs noatime,noexec,nodiratime 0 0" | sudo tee -a /etc/fstab

sudo mount /raid
```

Components should be started in the following sequence: mongod processes, config servers, and mongos routers. A shard server may consist of a mongod process or it may have a replica set of mongod processes.

Before launching any mongod processes, update /etc/hosts file and replace the DNS names of sharding components with their actual IP addresses. This file should be modified on the client, configuration servers, and all shards. Below is an example of /etc/hosts content:

```
127.0.0.1      localhost localhost.localdomain

192.168.1.1   rnd-nosql-server-master

192.168.1.2   rnd-nosql-server1

192.168.1.3   rnd-nosql-server2

192.168.1.4   rnd-nosql-server3

192.168.1.5   rnd-nosql-server4
```

Launch a mongod process on each of the nodes:

```
$ mongo-bin/mongod.sh
```

The --configsvr process should be run with a --configdb parameter defining the route to config database(s). Remember that a --configsvr process should be run for each config server while mongos can be run on any servers you choose. The DNS names should be used instead of IP addresses when defining the --configdb parameter:

```
$ mongo-bin/mongos.sh
```

When the shard components are started you should connect to the admin database. This can be done through mongos:

```
$ bin/mongo <mongos-hostname>:<mongos-port>/admin
```

Enter the JavaScript sharding commands in the admin console:

```
db.runCommand({

        addshard : "<serverhostname>[:<port>]"

});
```

After one or more shards have been added, sharding should be enabled in the database:

```
db.runCommand({

        enablesharding : "<dbname>"
});
```

To automate these steps, commands should be added to the mongo-bin/js/config-shards.js file which will start replay in the shell:

```
$ mongo-bin/config-shards.sh
```

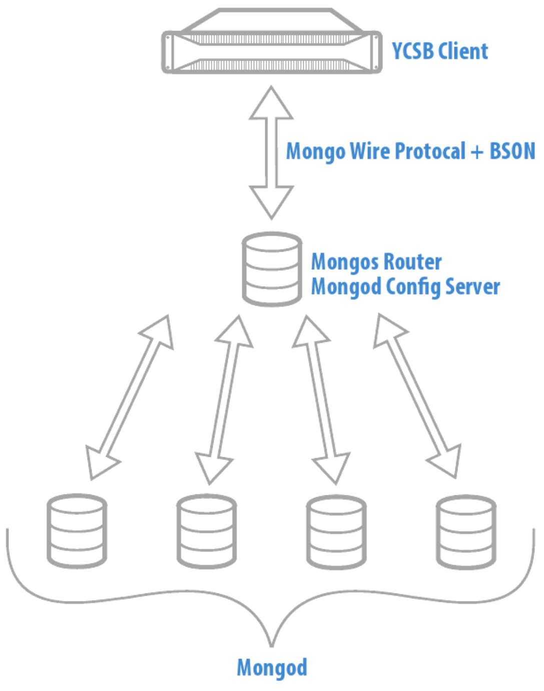
To verify that the cluster is running properly, do the following:

```
$ bin/mongo <mongos-hostname>:<mongos-port>/UserDatabase

> db.stats();
```

MongoDB cache uses memory-mapped files to work with data. The virtual size parameter displays the size of data mapped and Resident Bytes Counter tracks the size of data cached in RAM.

Resulting Cluster



7.4 Riak

General Information

History	In 2008, a group of engineers founded Basho Technologies and the work on a new-generation platform based on the principles stated in the Amazon's Dynamo paper began. Riak was designed as a highly-available open-source distributed

	database featuring linear scalability, low latency, and a decentralized architecture. Apart from the Riak database, the company offers a range of Riak-based products and services.
Notable Customers	AOL, Ask.com, Comcast, Mozilla, Yammer, Mobile Interactive Group, Wikia, Opscode
Best Use	Data-intensive e-commerce solutions, risk-free systems
Type	Riak is a key-value storage, which organizes data into buckets, keys, and values. Each key/value pair is stored in a bucket along with a unique key reference value.
Language	Erlang, C, C++, JavaScript
License	Apache License v 2.0
Commercial Support	Basho consulting services
Community Support	The riak-users@lists.basho.com mailing list, the #riak IRC room at irc.freenode.net, the technical blog at http://basho.com/blog/technical/, and the Twitter handle at http://twitter.com/#!/basho; for more channels visit http://wiki.basho.com/Community.html.

Technical Information

Language Bindings & Clients	Erlang, JavaScript, Java, PHP, Python, Ruby clients and community contributed projects for .NET, JavaScript, Python (and Twisted), Griffon, Perl, and Scala
Protocols	Native Erlang, RESTful HTTP, Protocol Buffers Client 0.10

Operations

Querying	The key/value model is used for accessing data. Javascript-based MapReduce is used to perform queries that exceed the existing limitations.
ACID, CAP	Riak focuses on availability and partition tolerance of the CAP theorem.
Secondary Indexes	Secondary indexes are available with the only index-capable backend ELevelDB as of version 1.0.
MapReduce	Riak's MapReduce consists of two components: a list of inputs and a list of "steps", or "phases" (map, reduce, link).

Initializing a Riak Cluster on Amazon EC2.

EC2 Security Group for Riak

Step 1: In My Resources section of your Amazon EC2 Console Dashboard, select Security Groups.

Step 2: Click Create Security Group. Fill out a description and click Yes.

Step 3: Click Inbound and add rules for the following ports.

Port	Rule Type	Description
22	SSH	Default SSH port
8087	Custom TCP Rule	Protocol buffers interface
8098	Custom TCP Rule	HTTP interface
8099	Custom TCP Rule	Intra cluster data handoff
0-65535	All TCP	Source should be the group id of this security group

Deploying an Amazon Image

On your Amazon EC2 Console Dashboard, click Launch Instance and the Request Instances Wizard will be automatically started. Create an Amazon Instance with the Riak Security Group, you specified above.

Managing an Amazon EC2 Disk

For more information on how to enlarge an ext3 Amazon EBS volume refer to Amazon EC2 Notes.doc

Installation Instruction.

Download Erlang

Riak has platform specific instructions on how to download a compatible version of Erlang. We used the most recent version of Erlang R15B. On Mac OS, execute the following commands:

```
$ curl http://www.erlang.org/download/otp_src_R15B.tar.gz > otp_src_R15B.tar.gz
$ tar -xzf otp_src_R15B.tar.gz

$ cd otp_src_R15B

$ ./configure

$ make

$ sudo make install
```

Erlang can be installed with a supported package manager:

```
$ sudo yum install erlang
```

Download Riak

Download the source code of Riak version 1.1.1:

```
$ wget http://downloads.basho.com/riak/riak-1.1.1/riak-1.1.1-1.el5.x86_64.rpm

$ sudo rpm -Uvh riak-1.1.1-1.el5.x86_64.rpm
```

Installation from source requires the following:

```
$ sudo yum install gcc gcc-c++ glibc-devel make openssl098e

$ wget http://downloads.basho.com/riak/riak-1.1.1/riak-1.1.1.tar.gz

$ tar zxvf riak-1.1.1.tar.gz

$ cd riak-1.1.1

$ make rel

$ sudo mv rel/riak/ /usr/lib/riak-1.1.1

$ cd .. ; rm -rf ./riak-1.1.1*
```

Specifying Riak Settings

Riak is a key/value store inspired by Amazon Dynamo. It is schema free and scales predictably and easily. Essentially, Riak is a decentralized key/value store with a MapReduce engine on board and an HTTP/JSON interface.

If Riak is installed from Source, app.config and vm.args configuration files will be located in etc/. In case of a binary installation, they can be found at /etc/riak. Such node attributes as search config, backend, system monitor, etc. can be assigned in the app.config file. Some parameters, such as cookie or name, can be passed to an Erlang node through the vm.args file.

Riak clients can access the cluster via HTTP or Protocol Buffers. Both these methods can be used at the same time. The IP address and TCP port should be configured in order to check for clients' connections. The default localhost address for HTTP is 8098 (cannot be accessed by other nodes), but it can be changed to enable clients to work via HTTP. Once the address for the Protocol Buffers is set to 0.0.0.0 and the address for the TCP port is set to 8087, clients can connect to any server address.

```
node[:riak][:core][:http] = [["127.0.0.1", 8098]]

node[:riak][:kv][:pb_ip] = "0.0.0.0"

node[:riak][:kv][:pb_port] = 8087
```

Storage backend and its specific options are tunable for each backend. These can be specified in the configuration. The possibility to specify different backends for different buckets is a crucial factor for range scan queries and it is implemented in Riak.

A range scan is performed by indexes and may not show such results as zero or one row. It is possible to use secondary indexes to do range scans in Riak. If you use other ways of scanning records, such as listing a bucket, this will heavily increase the load on the cluster. EleveIDB is currently the only storage backend that enables scanning by secondary indexes in Riak.

With the EleveIDB backend, when secondary indexes are enabled, a special index field called $key is implicitly added to all objects. For an application, this field marks the point where range queries are performed across the primary keys in a bucket.

Running a Cluster

Specifying Cluster Settings

A set of physical hosts is used to run a Riak cluster (one host for each Riak node). Every node in the cluster runs a set of virtual nodes called "vnodes". The allocated portions of the key space are stored across them.

In default settings, the number of partitions is defined by the “ring_creation_size” value, which shows the number of nodes in the cluster. The possibility to change this value gives much freedom to users, who can adjust the system to their custom needs. For improved performance, make sure that this value exceeds the number of nodes a few times. Note that the ring creation size number should be a power of 2 (i.e. 64, 128, 256, etc.).

The app.config template with specified properties is located in riak-1.1.1/riak-etc/. Before launching nodes in the cluster, the node name should be modified. This is a crucial setting for the Erlang VM, as communication and coordination of nodes in a Riak cluster is performed through node names.

The node name is configured in vm.args and its default value is 127.0.0.1 (a.k.a. localhost):

```
-name riak@127.0.0.1
```

When long host names are used, they are flagged with a -name mark, and the system makes sure that the host name part of the node name contains at least one period. In Riak, short and long names are processed similarly. The only difference is that short names are marked with the -sname flag.

You can change these values using, for example, the IP address or a resolvable host name for the particular node. A new name will be as follows:

```
-name riak@rnd-nosql-node1
```

Then launch the Riak daemon process and check whether the node is operating properly:

```
$ cd /usr/lib/riak-1.1.1/

$ bin/riak start

$ curl http://localhost:8098/stats
```

You should repeat this action for every node in your cluster.

When the node is active, it can be joined to an existing cluster by sending a join request from the new node to a random seed node:

```
$ bin/riak-admin join riak@rnd-nosql-node1
```

Check the current status of all cluster members:

```
$ bin/riak-admin member_status
```

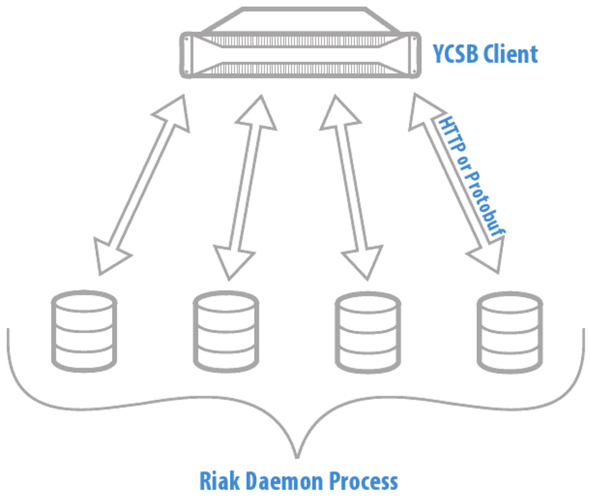
The output of the command will be as follows:

```
--------------------------------------------------------------------------
Status    Ring    Pending    Node
--------------------------------------------------------------------------
valid     25.0%     --       'riak@rnd-nosql-node1'
valid     25.0%     --       'riak@rnd-nosql-node2'
valid     25.0%     --       'riak@rnd-nosql-node3'
valid     25.0%     --       'riak@rnd-nosql-node4'
--------------------------------------------------------------------------
Valid:4 / Leaving:0 / Exiting:0 / Joining:0 / Down:0
```

Creating a Data Schema

Riak is a schema free database, which means that only a bucket name and an object identifier are required to import or load data.

Resulting Cluster Scheme



7.5 MySQL Cluster

General Information

History	MySQL Cluster is a technology that helps users of the MySQL database management system to leverage the advantages of the shared-nothing architecture. MySQL Cluster was first included in the MySQL 4.1 release in November 2004. This technology provides high availability, improved performance, and nearly linear scalability.
Notable Customers	SPEECH DESIGN GmbH, Telenor, Nokia, NEC, 8x8
Best Use	Traditionally, it is used in telecommunications network services, financial services, and online gaming.
Type	This is a relational shared-nothing in-memory database. All indexed columns are maintained in distributed memory. There are 2 scenarios for maintaining non-indexed columns. They can either be kept in distributed memory or on disk with an in-memory page cache.

Language	C, C++
License	MySQL Cluster is open-source software distributed under the GPL License. Commercial licenses are available for OEMs, ISVs, and VARs.
Commercial Support	MySQL Cluster CGE is a commercial edition of MySQL.
Community Support	Various mailing lists, forums, and dedicated channels on Internet Relay Chat (IRC)

Technical Information

Language Bindings & Clients	SQL
Protocols	—

Operations

Querying	SQL
ACID, CAP	MySQL Cluster is an ACID-compliant transactional database. A synchronous replication through a two-phase commit protocol is used to ensure that data is written to multiple nodes. Starting with version 5.1, MySQL Cluster allows for asynchronous replication between clusters. The algorithm for hashing the primary key of the table guarantees automatic partitioning of data across all nodes in the system. The foreign key construct is ignored.
Secondary Indexes	+
Map/Reduce	n/a

Installation

Download MySQL Cluster of the appropriate version and architecture for the platform you use from http://www.mysql.com/downloads/cluster/. Extract the archive and move the contents to a destination directory:

```
$ wget http://mysql.he.net/Downloads/MySQL-Cluster-7.2/mysql-cluster-gpl-7.2.5-linux2.6-x86_64.tar.gz

$ tar zxf mysql-cluster-gpl-7.2.5-linux2.6-x86_64.tar.gz

$ sudo mv mysql-cluster-gpl-7.2.5-linux2.6-x86_64 /usr/lib/mysql-cluster-7.2.5
$ rm -f mysql-cluster-gpl-7.2.5-linux2.6-x86_64.tar.gz
```

Refer to the Amazon EC2 notes document for instructions on how to create a RAID array and data directories for the mysqld database process:

```
sudo chown ec2-user /raid/var/lib

mkdir -p /raid/var/lib/mysql-cluster/mysqld /raid/var/lib/mysql-cluster/ndb

sudo ln -s /raid/var/lib/mysql-cluster/mysqld /var/lib/mysql-cluster/mysqld

sudo ln -s /raid/var/lib/mysql-cluster/ndb /var/lib/mysql-cluster/ndb

sudo chown ec2-user /var/lib/mysql-cluster/mysqld /var/lib/mysql-cluster/ndb
```

There are three types of processes implemented in MySQL Cluster:

ndb_mgmd–this process is run on the management node and is used for cluster configuration and monitoring.

ndbd/ndbmtd–this process stores data on a data node.

mysqld–this process is run on an SQL node; it is a server process that connects to all data nodes of the system to store and retrieve data. This type of node is optional and data can be queried directly through NDB API.

MySQL Cluster uses 2 types of configuration files. Configurations used with Amazon EC2 Instances can be found in the folder mysql-cluster/mysql-cluster-conf:

config.ini is a global configuration file, which is only read by the management server of MySQL Cluster. Then, the information contained in config.ini is spread by the server to all processes united in the cluster. The description of all nodes organized in the cluster is also stored in the config.ini file.

my.cnf specifies options for all MySQL Cluster executables.

You should pay attention to the following parameters specified in config.ini:

DataMemory–this parameter sets the amount of space (in bytes) allocated for storing database records. Both actual records and ordered indexes are stored in memory allocated by DataMemory. Primary key and unique hash indexes use IndexMemory. However, when a primary key or a unique hash index is created, an ordered index is created on the same keys, unless otherwise is specified in the index creation statement. DataMemory also contains undo information. An unchanged copy of every record is saved to this memory space after an update. The default DataMemory size is 80M.

IndexMemory is used for storing hash indexes created for an NDBCLUSTER table. Hash indexes are usually assigned to primary key indexes, unique indexes, and unique constraints. Unlike updates that do not affect the amount of index memory used, inserts require more memory–about 21–25 bytes per record for every index.

DiskPageBufferMemory specifies the amount of space on disk used for caching pages. This parameter is defined in bytes and is set in the [ndbd] or [ndbd default] section of the config.ini file. DiskPageBufferMemory = 0.8 x ([total memory] - ([operating system memory] + [buffer memory] + DataMemory + IndexMemory))

The maximum possible number of operations performed per transaction is defined by the following parameters: MaxNoOfLocalOperations and MaxNoOfConcurrentOperations.

Run the ndb_mgmd process:

```
$ mysql-cluster-bin/mysql-ndb-mgmd.sh
```

Then execute an ndb database process on each of the shards:

```
$ mysql-cluster-bin/mysql-ndbd.sh
```

Finally, launch the mysqld process on the same nodes, where you launched the ndb processes:

```
$ mysql-cluster-bin/mysqld.sh
```

Connect to the mysqld process through the client and create a database and a table for testing. You will find a DDL script for creating a schema in the following file:

mysql-cluster/src/main/resources/create_database.sql

```
create table user_table(
  ycsb_key varchar(32) primary key,
  field1 varchar(100), field2 varchar(100), field3 varchar(100), field4 varchar(100),
  field5 varchar(100), field6 varchar(100), field7 varchar(100), field8 varchar(100),
  field9 varchar(100), field10 varchar(100))
  max_rows=1000000000 engine=ndbcluster partition by key(ycsb_key);
```

MAX_ROWS is a per-partition option defining the maximum number of rows that can be stored in a table. In general, multiply the number of rows you're expecting to use by two and you'll get the optimal value for the MAX_ROWS parameter. The limit for the MAX_ROWS value is 4294967295; all larger values will be set back to this limit. This is a maximum value for the NDB storage engine.

MySQL Cluster is recognized as an in-memory database, but it can also store data on disk. Not all data in a disk table is actually stored on disk. The first 256 bytes of a BLOB/TEXT column and index columns are not (the whole indexed column is kept in the main memory, not just the index).

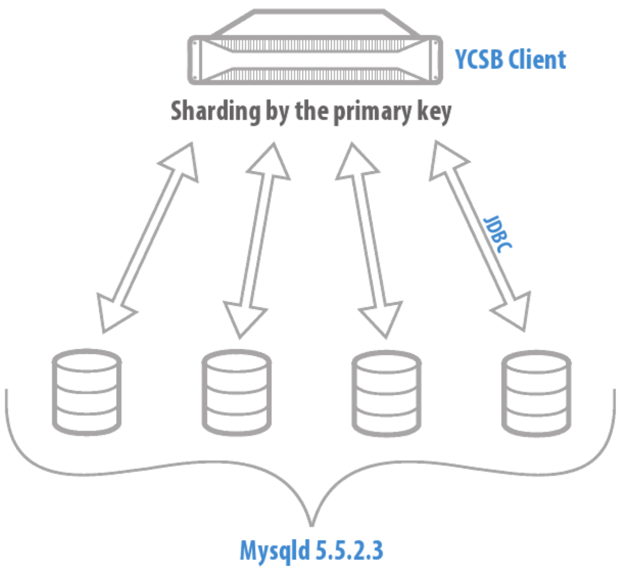
It's important, that varchar and varbinary types can vary in size. This means they will only occupy the amount of space inserted into them.

The table above was altered to use disk storage. Rows are persisted to tablespace, which acts as a container for disk data records while undo information required for rolling transactions back is stored in undo log files.

The **initial_size parameter** sets the total size of the data file in bytes. The size of the file cannot be changed after it has been created, but new data files can be added to the tablespace, if you follow the instructions for altering tablespace.

The **undo_buffer_size** is taken out of SharedGlobalMemory. As a rule, the size of SharedGlobalMemory should be increased to fit the undo buffer. An undo buffer of 128M in size will be enough for most applications, but you should keep about 384M for the SharedMemoryGlobal option.

Resulting Cluster Scheme



8. Appendix C

8.1 Amazon EC2 Notes

Enlarge an ext3 Amazon EBS volume

1. Do the following at the EC2 Web Console:

   1.1. Unmount the volume.

   1.2. Take a snapshot of the volume.

   1.3. Create a new larger volume from the snapshot.

   1.4. Detach the old volume

   1.5. Attach the new larger volume.

2. The following actions must be performed once for each instance:

   2.1 In the shell of an instance type the following:

```
sudo resize2fs /dev/sda1
```

   2.2 Check image size:

```
df -h
```

   2.3 Edit the sudoers file:

```
Sudo nano  /etc/sudoers
```

   2.4 Conmment line

```
Defaults    requiretty
```

Changing hosts file

There are some scripts for automating the process of changing the hosts file.

```
sudo nano /etc/hosts
```

You need two times the total capacity and each disk has to equal 2*(capacity)/(num disks).

Creating RAID10 (RAID0 mirroring of RAID1 stripping)

1. Define the X.509 certificate and private key:

```
export EC2_PRIVATE_KEY=`ls ~/X.509/pk-6L3PG43XHLGMHIJOBABBWJFWP23SU2CY.pem`

export EC2_CERT=`ls ~/X.509/cert-6L3PG43XHLGMHIJOBABBWJFWP23SU2CY.pem`
```

Instance id and availability zone will be retrieved and defined as follows:

```
export EC2_INSTANCE_ID=`wget -q -O - http://169.254.169.254/latest/meta-data/instance-id`
```

```
export EC2_AVAILABILITY_ZONE=`wget -q -O - http://169.254.169.254/latest/meta-data/placement/availability-zone`
```

2. Create volumes in the availability zone $EC2_AVAILABILITY_ZONE and then attach them to the instance $EC2_INSTANCE_ID:

```
for x in {1..4}; do \
  ec2-create-volume --size 50 --availability-zone $EC2_AVAILABILITY_ZONE; \
done > ~/ec2-volumes
i=0; \
for vol in $(awk '{print $2}' ~/ec2-volumes); do \
  i=$(( i + 1 )); \
  if [ -z "$DEVICES" ]; then export DEVICES="/dev/sdh${i}"; else export DEVICES="$DEVICES /dev/sdh${i}"; fi; \
  ec2-attach-volume $vol -i $EC2_INSTANCE_ID -d /dev/sdh${i}; \
done
```

3. Install the RAID driver and the xfs file system:

```
sudo yum install mdadm xfsprogs
```

4. Then, create a new RAID 0 or RAID 10 volume from disks. According to some articles floating around, the best settings for an EBS drive use the chunk size of 256:

```
sudo mdadm --create /dev/md0 --level=raid0 --metadata=1.2 --chunk=256 --raid-devices=4 /dev/sdh*
```

4.1. Prior to the creation or usage of any RAID devices, the /proc/mdstat file shows no active RAID devices. Once you create a RAID device, verify RAID by checking the config file with mdadm:

```
cat /proc/mdstat
sudo mdadm --detail /dev/md0
```

5. Set read ahead buffer to 64K:

```
blockdev --setra 65536 /dev/md0
```

6. Next, we need to add the devices to the mdadm.conf file, so the drive configuration is persisted:

```
echo DEVICE $DEVICES | sudo tee /etc/mdadm/mdadm.conf

sudo mdadm --detail --scan | sudo tee -a /etc/mdadm.conf


sudo mkdir /etc/mdadm/

sudo mdadm -Es | sudo tee /etc/mdadm/mdadm.conf
```

7. Construct the xfs file system on the /dev/md0 drive:

```
sudo mkfs.xfs /dev/md0
```

8. Activate auto-mounting on reboot:

```
echo "/dev/md0 /raid xfs noatime,noexec,nodiratime 0 0" | sudo tee -a /etc/fstab
```

noexec disables scripts execution and reduces the probability of falling to a scripted attack

noatime will not update access time on the files

nodiratime will not update the atime field of the directory

noexec

9. Mount and format the drive:

```
sudo mkdir /raid

sudo mount /raid
```

10. Verify that the drive is in operation:

```
df -h /raid
```

The procedure of configuring drives for RAID initializes them by writing to every drive location. When configuring software-based RAID, make sure to change the minimum reconstruction speed to increase the speed of the initial construction:

```
echo $((30*1024)) > /proc/sys/dev/raid/speed_limit_min
```

11. To remove the drive, run the following commands:

```
sudo umount /raid

sudo mdadm --stop /dev/md127

sudo mdadm --remove /dev/md127

export EC2_PRIVATE_KEY=`ls ~/X.509/pk-6L3PG43XHLGMHIJOBABBWJFWP23SU2CY.pem`

export EC2_CERT=`ls ~/X.509/cert-6L3PG43XHLGMHIJOBABBWJFWP23SU2CY.pem`

export EC2_INSTANCE_ID=`wget -q -O - http://169.254.169.254/latest/meta-data/instance-id`
export EC2_AVAILABILITY_ZONE=`wget -q -O - http://169.254.169.254/latest/meta-data/placement/availability-zone`

export EC2_REGION=`curl http://169.254.169.254/latest/dynamic/instance-identity/document|grep region|awk -F\" '{print $4}'`

i=0; \

for vol in $(awk '{print $2}' ~/ec2-volumes); do \

  i=$(( i + 1 )); \

  ec2-detach-volume $vol -i $EC2_INSTANCE_ID -d /dev/sdh${i} -f; \

done

sleep 30

for vol in $(awk '{print $2}' ~/ec2-volumes); do \

  ec2-delete-volume $vol –-region $EC2_REGION; \

done
```

8.2 YCSB Configuration

Client Configuration

Creating an Amazon EC2 Instance:

Client node is used to run the Yahoo Cloud Serving Benchmarking workloads. The node is deployed on an Amazon High-CPU Extra Large Instance, which is a 64-bit platform with 7 GB of memory, 20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each) and 8GB of instance storage. Instance rnd-nosql-sq Security Group and SSH key rnd-nosql-key were created for the client.

1. Logging Into an Amazon EC2 Instance

1.1. Login to Amazon Instance as ec2-user with the SSH connectivity tool:

ssh -i rnd-nosql-key.pem ec2-user@<instance>.amazonaws.com

1.2. Use sudo to get root access (if required):

sudo sh or *sudo su*.

2. Managing Amazon EC2 Disk

2.1. To enlarge an ext3 Amazon EBS volume, refer to *Amazon EC2 Notes.doc*

3. Installing Java

3.1. Configure the alternatives system and plug-ins. If java has already been installed, skip this section:

alternatives --config java

3.2. Download the appropriate version of Java:

curl http://download.oracle.com/otn-pub/java/jdk/{version}-rpm.bin > jdk-{version}-rpm.bin

3.3. Change the file's permission with chmod:

chmod 755 ./jdk-{version}-rpm.bin

3.4. Run the file using the following command:

jdk-{version}-rpm.bin

3.5. Export the JAVA_HOME variable to the user's .bash_profile:

export JAVA_HOME=/usr/java/jdk-{version}

export PATH=$PATH:$JAVA_HOME/bin

4. Installing and Configuring Apache Maven

4.1. Find the download URL on the Apache maven download page:

http://maven.apache.org/download.html

4.2. Change the permission to 700 (yes, only you should be authorized to do it):

chmod 700 apache-maven-{version}-bin.tar.gz

4.3. Extract the files from the above archive. The command below will create the apache-maven-{version} directory in your current directory:

tar xzf apache-maven-{version}-bin.tar.gz

4.4. Move the above directory to the /usr/local directory using the following command:

mv ./apache-maven-{version} /usr/local

4.5. Create a symbolic link for maven:

cd /usr/local

ln -s ./apache-maven-{version} apache-maven

4.6. Add maven to your system path, so that you can call it from anywhere. Open /home/ec2-user/.bash_profile file:

*vi ~/.*bash_profile

4.7. Add the following to the end of the file:

export M2_HOME=/usr/local/apache-maven

export PATH=$PATH:$M2_HOME/bin

4.8. Save the changes by pressing the escape key and then type :wq!.

4.9. Refresh the bash profile (logout & login) and check the maven version by entering the following:

mvn –version

5. Installing Git and Assembling a Project (If Required)

5.1. Install the required dependencies:

yum -y install zlib-devel openssl-devel cpio expat-devel gettext-devel gcc

5.2. Download the Git distribution:

wget http://git-core.googlecode.com/files/git-1.7.8.tar.gz

5.3. Extract the downloaded archive and change the path to the extracted directory:

tar -xzvf ./git-1.7.8.tar.gz

cd ./git-1.7.8

5.4. Compile and install Git:

make

make install

5.5. Check the existing keys:

ls -al ~/.ssh

5.6. Generate a private-public key pair, if required:

ssh-keygen

5.7. Add the public key to GitHub:

less ~/.ssh/id_rsa.pub

5.8. Clone the NoSQL-research project from GitHub:

git clone git@github.com:Altoros/NoSQL-research.git

5.9. Change the current directory to:

cd ./NoSQL-research/src/

5.10. Change the permission for the checked out folder:

sudo chmod -R 777 ./NoSQL-research

5.11. Make a project:

mvn clean install