The NoSQL Technical Comparison Report

Cassandra (DataStax), MongoDB, and Couchbase Server

www.altoros.com
twitter.com/altoros

By Altoros
September 2014

Table of contents

1. Report overview

This report provides an in-depth analysis of the leading NoSQL systems: Cassandra, Couchbase Server, and MongoDB. Unlike other NoSQL comparisons that focus only on one or two dimensions, this research approaches the evaluated solutions from different angles to help you choose the best option based on performance, availability, ease of installation and maintenance, data consistency, fault tolerance, replication, recovery, scalability, and other criteria.

The appendices contain information on configuring the benchmark environment, as well as performance and scalability charts. These charts demonstrate how MongoDB v2.6.1, Cassandra v2.0.8 (DataStax Enterprise), and Couchbase Server v2.5.1 performed and scaled under different types of workloads (update, delete, insert, and read). When measuring performance, we used two data sets of 50 million and 3 million records to discover how database behavior changes depending on the amount of data.

In addition to general information on each evaluated data store, the report contains recommendations on the best ways to configure, install, and use NoSQL databases depending on their specific features. In chapter four, you will find a comparative table that summarizes how MongoDB, Cassandra, and Couchbase Server scored for each criterion—on a scale from 1 to 10.

A note on methodology: For criteria based on measurable performance data, scores were applied based solely on the benchmark results of the three products under evaluation. Details about the specific benchmark environment are provided in Appendix A. For criteria not based on performance benchmarks—e.g., installation and maintenance procedure—scores were applied based on factors such as ease of use, effort required, and completeness of functionality.

General information about the analyzed databases

Cassandra is a column-family store operating under the guidance of the Apache Software Foundation. Initially developed by Facebook, it aims to provide availability and scalability. With its flexible consistency models, the architecture of Cassandra is a good fit for write-intensive applications. In this benchmark, we used the DataStax Enterprise edition.

MongoDB is a document-oriented NoSQL database. It has extensive support for a variety of secondary indices, strong features around documenting, and a special approach to scaling and durability.

Couchbase Server is both a document-oriented and a key-value NoSQL system. It guarantees high performance with a built-in object-level cache, asynchronous replication, and data persistence.

2. Installation and configuration

This section is dedicated to the ease and convenience of installation and configuration.

Cassandra (DataStax)

The common way to install Cassandra on CentOS 6.5 is using the Yum Package Manager and DataStax OpsCenter, a visual management and monitoring solution for Apache Cassandra and DataStax Enterprise. To install Cassandra, you need to do the following:

1) Edit the Yum repository specification on a single node.
2) Install the OpsCenter package using Yum.
3) Check the OpsCenter IP/hostname settings.
4) Create a new cluster using the Web UI provided by the installed OpsCenter, specify the list of nodes where Cassandra will be installed, and identify the sudo credentials.
5) Customize cluster configuration.

The last step involves adjusting general settings for operating directories, your network, and ports; configuring caches, memory management, and compaction; as well as selecting the manner in which the data will be distributed and the number of reading and writing threads.

Couchbase Server

Couchbase Server installation has the following basic steps:

1) Download and install the Couchbase Server package on each of the cluster nodes.
2) Configure a single node as a standalone cluster via CLI, REST API, or a Web UI.
3) Add all the other nodes to the cluster.

You can find a more detailed description of the installation procedure in the Couchbase documentation for CentOS, RHEL, Ubuntu, Windows, and Mac OS. Windows and Mac OS are supported only for developers' use.

Reference: http://docs.couchbase.com/couchbase-manual-2.5/cb-install/

MongoDB

To run MongoDB on CentOS 6.5 with Yum:

1) Install MongoDB on each node of the replica set.
2) Deploy the replica set.

There is a list of recommended settings that should be used for deployment, including limits for user resources and swap usage. As far as memory utilization is concerned, the more RAM a node has, the better read performance it provides, since a database may have a larger size of cache and the disk I/O for reads is reduced. Engineers who are taking part in deployment planning and implementation should be aware of anti-patterns, which are ineffective and/or counterproductive. Some examples of such patterns are as follows: using network attached storage, shared network file systems, exceeding heap space size, etc.

Summary

Thanks to Couchbase's integrated admin console, everything can be configured in one place, which makes it easier to install. MongoDB is based on the master-slave principle that makes deployment more complicated than in peer-to-peer data stores. Cassandra has master-less architecture where all nodes in the cluster are equal. It can be installed using a package manager.

Table 1. Ease and convenience of installation on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
6	10	9

3. Comparison of functionality and structure

3.1. Architecture

3.1.1. Topology

Cluster topology refers to the arrangement of system components, node roles, and communication patterns. Topologies can be fixed or flexible. Different cluster topologies lead to different internal and external data flows, system scalability, and availability.

MongoDB

MongoDB supports master-slave replication and a variation of master-slave replication, known as replica sets. If a master goes down, one of the replicas becomes a new master node and continues to accept write operations. Several replica sets could be combined in a sharded cluster. A sharded cluster requires additional configuration servers, which hold DB metadata and a process that distributes requests between shards of a cluster. As mentioned above, MongoDB's master-slave concept is generally more complicated in deployment than peer-to-peer databases.

Couchbase Server

All nodes in a Couchbase Server cluster are equal in roles and sizes. Each node runs two major components: Cluster Manager and Data Manager.

Cluster Manager is responsible for node configuration and communication, cluster monitoring, keeping the smart client up-to-date, and a number of other activities. The smart client has a cluster map and connects directly to the cluster node, which holds an active data chunk called vBucket. (Couchbase defines the term vBucket as the “owner” of a subset of the key space of a cluster. Note that it is not the same as a bucket, which is a multi-tenancy unit that represents a virtual Couchbase Server instance inside a cluster.) In the event of a server failure, the Cluster Manager detects that a node is down, promotes a live vBucket replica to the active status, and updates the cluster map. When a node is added or removed, the Cluster Manager is responsible for data rebalancing.

Data Manager takes care of data storage and access. It consists of a storage engine, a query engine, and object-managed cache. Multi-threaded object-managed cache provides consistent low latency for reads and writes by keeping frequently used documents’ metadata and content in memory. The storage engine automatically migrates the least recently used data from RAM down to SSDs and spinning hard drives. This engine has an append-only design, so any document mutation goes to the end of a file. Multiple files are kept open at the same time. When the ratio of the actual data to the current file size goes below a certain threshold, a compaction process is triggered. The Rack Awareness feature added in Couchbase Server 3.0 allows for logical grouping of servers by racks or availability zones. The query engine builds distributed primary and secondary indexes.

The listed Couchbase Server components run locally on each of the cluster nodes. Some of the sub-components are promoted to be cluster-wide services, which are enabled on one node at a time. The node running a cluster-wide service is chosen by an election mechanism.

Notes from an architect’s point of view:

All nodes in a cluster are equal. No single point of failure. All configurations can be done via an integrated admin console. In addition, the XDCR topology is rather flexible (one-to-many, many-to-one, and many-to-many).

Cassandra (DataStax)

Cassandra is a cluster ring with symmetric nodes without any masters. Each node is assigned a unique token (or a set of tokens with *vnodes*) that determines a range of keys for which this replica will be the first.

Cassandra is able to efficiently route requests according to the network topology and distribute replicas by grouping machines into data centers and racks. Network topology is described by a snitch. It tells Cassandra which racks and data centers are written to and read from. SimpleSnitch, which is used by default, does not recognize data center or rack information. Unlike it, a dynamic snitch monitors the health of reads and finds the best suitable location for them.

Summary

Cassandra and Couchbase topologies are similar: the nodes in a cluster are homogenous; cluster-wide services are hidden from clients and administrators most of the time; and clusters are easy to scale. However, Couchbase’s XDCR topology is much more flexible with several options (one-to-many, many-to-one, or many-to-many). The integrated admin console also makes it somewhat simpler. Cassandra considers all nodes to be part of a single cluster, even if they are spread across multiple data centers. MongoDB's architecture is less encapsulated and more complicated without any significant benefits.

Table 2. Topology of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
7	10	9

3.1.2. Consistency

Refers to the data consistency model that specifies a contract between a programmer and a system, which guarantees that results of concurrent database operations will be predictable.

MongoDB

By default, the primary MongoDB node is targeted with all the reads and writes, which means that data is fully consistent. However, if read operations are permitted on secondary nodes, MongoDB guarantees only eventual consistency. So, developers and architects should take into account that there is a delay when data is replicated from master to slave nodes.

Couchbase Server

A single Couchbase data partition with its own key space is called a "bucket." Each bucket is split into a number of vBuckets. Each vBucket is replicated separately and only a single replica is active at the same time. The active replica serves all the requests to the subset of keys that the vBucket owns. This guarantees the system's strong inter-cluster consistency. The cross-datacenter replication (XDCR) is eventually consistent thanks to an asynchronous mechanism.

Reference:
http://info.couchbase.com/rs/northscale/images/Couchbase_WP_Cross_Datacenter_Replication_in_Couchbase_Server.pdf

Notes from a developer's point of view:

One can choose between the "persisted to" option (the operation is completed when data is persisted to the specified number of nodes) and the "replicated to" option (the operation is completed when data is replicated to the specified number of nodes) for each call. The observe method provides a way to get the actual number of replicated and persisted nodes, as well as acknowledge if the key has been indexed.

Cassandra (DataStax)

In addition to regular eventual consistency, Cassandra supports flexible consistency levels. The client application identifies to what extent a row of data is up-to-date and how much of it has been synchronized—compared to other available replicas.

On the weakest consistency level, a write operation will be completed if it is accepted by at least one node. On the level of a replica set, a write operation must be approved by a quorum (the majority of replica nodes). You can choose among a local quorum, each quorum, and all levels, which would provide the highest degree of consistency, but the lowest level of availability. So, although consistency with quorums is supported, there is a performance tradeoff for that.

Notes from a developer's point of view:

It is possible to set a required consistency level for any read (SELECT) or write (INSERT, UPDATE, DELETE, BATCH) operation via Thrift and CQL interfaces. In CQL, consistency specification is made up of USING CONSISTENCY keywords, followed by a consistency level identifier. The default consistency level is ONE.

Summary

Tunable consistency is available in each of the three databases. However, Cassandra is optimized for eventual consistency. Although it supports consistency with quorums, there is a performance tradeoff for that.

Table 3. Consistency of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	9

3.1.3. Fault tolerance (and partition tolerance)

Fault tolerance is the ability of a system to continue operating properly in case some of its components fail. Each of the databases has its own fault tolerance policy.

MongoDB

In terms of Brewer's CAP theorem, MongoDB is focused on consistency and partition tolerance. MongoDB employs master-slave architecture with failover, which enables a secondary member of the system to become a primary node, if the current primary node is unavailable. If all of the MongoDB configuration servers or router processes (mongos) become unavailable, the system stops responding. Compared to the systems with the peer-to-peer architecture, MongoDB is less fault-tolerant.

Couchbase Server

In terms of the CAP theorem, Couchbase Server is CP. It is a document-oriented database and each document is associated with a key. The entire key space is divided into 1,024 vBuckets. They are balanced across cluster servers and replicated according to replication factor settings specified for a particular bucket. A master vBucket replica is kept in the active state; replication is an asynchronous process. In the event of a server failure, a "failover" process must be triggered to transfer the master status for all of the vBuckets previously mastered on that server to other servers that have replica copies of those vBuckets. The "failover" can be manual or automatic.

If a cluster experiences more node failures than the number of replicas configured, and no rebalance operation is issued, data can be lost and/or become unavailable.

Cassandra (DataStax)

Partition tolerance is an integral part of the Cassandra architecture. Basically, the partitioner determines a node where data should be stored. The partitioner can distribute data evenly across the cluster based on the MD5 or Murmur3 hash—which is a recommended practice—or keep data lexically ordered by key bytes. The number of data copies is determined by the replica placement strategy. In addition, the cluster topology describes distribution of the nodes on racks and specifies the ideal number of data centers to use the network in a more efficient way.

Cassandra has order-preserving or byte-ordering partitioners that operate on partition key bytes lexicographically. However, they are not recommended for production deployments, since they can generate hot spots—a situation when some partitions close to each other get more data and activity than others.

Couchbase and Cassandra have architectures with no single point of failure. All nodes are the same and communicate based on the peer-to-peer principle, where user data is distributed and replicated among all nodes in the cluster. Replication ensures data accessibility and fault tolerance by storing copies of data on multiple nodes.

Summary

Although MongoDB is highly fault tolerant, it has a single point of failure. The cluster may stop responding, if a configuration server or a router process fails and there are no secondary instances available to replace it.

Table 4. Fault tolerance of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
7	10	10

3.1.4. Structure and format

The storage and index data structures are directly responsible for write, read, and space amplification. The amplification factor refers to the amount of work done physically to handle a single logical change. For write and read operations, that is RAM + disk + network hits per a single insert or read query respectively.

Data format is an embedded or external schema that determines how a single entry is treated. It may have the following values: a row, a document, a key and a binary value, etc.

MongoDB

MongoDB structures data into collections of BSON documents. BSON is short for Binary JSON, a binary-encoded serialization of JSON-like documents. Like JSON, BSON supports embedding documents and arrays within other documents and arrays.

Every document in MongoDB is stored in a record, which contains the document itself and extra space (or padding) that enables the document to grow along with updates. All of the records are contiguously located on a disk, and when a document becomes larger than the allocated record, MongoDB must allocate a new record. New allocations require MongoDB to move the document and update all indexes that refer to it, which increases the amount of time for updates and leads to storage fragmentation. Because of this fragmentation, MongoDB has significant storage overhead. B-trees are used for indexes.

Couchbase Server

In this system, data is written to a data file in the append-only manner. Couchbase Server uses multiple files for storage: a data file per partition, index files, and master files.

Data files are organized as B-trees. The root nodes contain pointers to intermediate nodes. These intermediate nodes contain pointers to leaf nodes. The root and intermediate nodes also track the sizes of documents under their sub-tree. The leaf nodes store the document ID, document metadata, and pointers to document content.

To prevent data files from growing too large and eventually exhausting the entire disk space, Couchbase periodically cleans up stale data from the append-only storage. It calculates the ratio of the actual data size to the current file size using the information stored in the B-trees. If this ratio goes below a certain threshold (configurable through the admin UI), a process called compaction is triggered.

Compaction scans through the current data file and copies non-stale data into a new data file. Since it is an online operation, Couchbase can still process requests. When the compaction process is completed, Couchbase copies the data that has been modified since the start of the compaction process to a new data file. The old data file is then deleted and the new data file is used until the next compaction cycle.

Couchbase Server has MapReduce-based indexes incrementally updated as data changes. To keep track of the indexed data partitions, it uses a tree structure called "B-Superstar."

Cassandra (DataStax)

Cassandra is a column-oriented storage of rows, where rows are organized into tables with a required primary key. A primary key consists of a mandatory partition key; rows can also be clustered by remaining columns (if it is required) of the primary key within a partition. Other columns may be indexed independently of the primary key with secondary indexes. Internally, secondary indexes are implemented as a hidden table, separate from the table that contains the values being indexed.

When a row is inserted into a table, it is first written to the commit log for recovery purposes and then to a per-table structure called Memtable. When the Memtable is full, it is flushed to the disk as an SSTable. Once flushed, no further writes may be done to this SSTable. This data structure is called LSM-tree (log-structured merge-tree); the implementation is called SAMT (sorted array merge-tree).

Notes from a developer's point of view:

It would be nice, if Cassandra also enabled an integrated cache. That would considerably improve read throughput and latency.

Summary

All the three data stores scored similarly in this category. Cassandra, however, could benefit from enabling an integrated cache that would improve throughput and latency. For instance, MongoDB employs memory-mapped files and pairs the CoW B-Tree with an in-memory hash table for the same purpose.

Note that the effectiveness of a NoSQL solution depends on how suitable the underlying architectural concepts are for each particular workload.

Table 5. The structure and format of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	9

3.2. Administration

3.2.1. Logging and Statistics

This section describes logging for performance, errors, and statistical analysis.

MongoDB

The standard approach of MongoDB to archiving includes archiving the current log file and starting a new one. You may also configure MongoDB to send log data to a syslog. MongoDB's logs hold all the information necessary for audit and control.

Couchbase Server

Couchbase Server supports Log4J, SLF4J, and SunLogger. Application and administration audit logging are planned features.

Cassandra (DataStax)

Out-of-the-box, audit logging and statistics are available in the DataStax Enterprise version. They are implemented as a log4j-based integration. It is possible to select the categories of audit events that should be logged and specify operations against any specific keyspaces that should be excluded/included from/into audit logging. The audit logs include a text description of an operation, information about the requestor, a node address, a category of the event, a keyspace, and a column family.

Notes from a developer's point of view:

The audit logs are a simple set of pipe-delimited name/value pairs written to a file by log4j. Any sort of processing can be built on top of this plain format.

Summary

Overall, MongoDB provides fewer logging options than Couchbase Server and Cassandra.

Table 6. Logging and statistics in NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
7	9	9

3.2.2. Configuration management

MongoDB

Initial MongoDB configuration can be done via the *mongod.conf* file or the command line parameters. The command line parameters are useful if MongoDB deployment is automated. The CLI tool is used to set configuration of a replica set or a sharded cluster. Additionally, there are external tools with a UI that can help to configure a cluster.

Couchbase Server

You can use the Web administration UI or CLI to configure Couchbase. It is easy to configure the whole cluster from the Web UI by accessing the port 8091 (by default) of any node. All nodes are expected to be equal in computational and storage capabilities. Replication and cross-region replication are configured per bucket.

Notes from a developer's point of view:

Administrative tasks can be automated using REST API.

Cassandra (DataStax)

The *cassandra.yaml* file is the main configuration file for Cassandra. DataStax simplifies configuration management through OpsCenter, which makes it possible to apply changes to a particular node or to the entire cluster.

Notes from a developer's point of view:

Cluster configuration can be managed manually by editing the *cassandra.yaml* file or through the Web GUI of OpsCenter.

Summary

Couchbase has a native Web UI for configuring the entire system. The DataStax version of Cassandra has a tool that partially automates configuration, but you still might need to edit the configuration file or use the command line. MongoDB can only be configured with the CLI or by editing the *mongod.conf* file. Those who are less experienced with this database can use third-party tools for configuring MongoDB.

Table 7. Configuration and management of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
7	10	9

3.2.3. Backup

Backup refers to the means of copying and archiving data. Backups are used to restore the original data after a loss event.

MongoDB

This database supports several methods for making a backup and restoring data in a MongoDB instance:

- Backup and Restore with Filesystem Snapshots (LVM system tool)
- MongoDB Tools (mongodump and mongorestore)

File system snapshots are reliable and can be created quickly, but you have to do more changes to the configuration outside the data store. MongoDB Tools are also quite good, but you will need more time to complete backup and restoration using them. Additionally, you can use MongoDB Management Service for backups, but there is a fee.

There is no built-in support for incremental backup, which requires backing up a full data snapshot every time. However, the MongoDB Management Service could be used for incremental backups.

Couchbase Server

This system has live backup and non-live restore utilities out-of-the-box. It is possible to backup a single bucket on a single node, all of the buckets on a single node, a single bucket from an entire cluster, or all of the buckets from the entire cluster. You can also backup cluster data by file copying.

Built-in incremental backup (which is already available in Cassandra) and restore utilities are planned for Couchbase Server v3.0.

Cassandra (DataStax)

Cassandra comes with the *nodetool* utility. It allows for backing up data by executing a snapshot command and creating a copy—actually, a hard link to an immutable SSTable—of on-disk data files stored in the data directory. It is possible to request a snapshot of all keyspaces, of a particular keyspace, or of a single column family. Snapshots are created on a particular node. To take a snapshot of the entire cluster, a parallel ssh tool (such as pssh) can be used.

Cassandra supports additional incremental backups, which allows for creating backups continuously without making the entire snapshot.

Notes from a developer's point of view:

Backups help you to recover data if client applications make errors when editing data. Cassandra stores backups locally and scheduled backups are not automated. If you would like to create backups regularly and automatically or store your backups outside the local storage, you should use a third-party sys admin tool (such as Bash, Puppet, etc.) on top of native backup utilities.

Summary

The latest versions of the solutions have very similar functionality. However, Couchbase and MongoDB do not support incremental backup.

Table 8. Backup of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
9	9	10

3.2.4. Disaster recovery

This section describes policies and procedures that enable recovery of the databases after global systems disasters.

MongoDB

If MongoDB does not shut down correctly, data files are likely to reflect an inconsistent state, which could lead to data corruption. Durability journaling is a feature that helps to eliminate the consequences of incorrect shutdowns. By default, MongoDB writes data to the journal every 100 ms. In this way, MongoDB can always recover to a consistent state, even in case of an unclean shutdown due to power loss or other system failure. To prevent data inconsistency and corruption, always stop the database in a proper way and use durability journaling.

Couchbase Server

This system supports XDCR replication and allows for restoring from backup with the Couchbase CLI utilities, as well as restoring from backed-up data directories. Couchbase Server 3.0 features include incremental backup and restore.

Cassandra (DataStax)

Cassandra provides several methods for backing up and restoring data. The first approach is based on using snapshots and incremental backups. Restoring a keyspace or a table from a snapshot is a straightforward operation. It is also possible to restore data from available replicas using the nodetool repair command. The repair process is intended for fixing inconsistencies across all of the replicas by pulling in data from the remaining nodes of the cluster.

Summary

Couchbase Server and MongoDB do not support incremental restore, yet. In addition, you have to use durability journaling with MongoDB, otherwise an unclean shut down may corrupt data.

Table 9. Disaster recovery of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
8	9	10

3.2.5. Maintenance

Typical maintenance tasks for a running cluster include adding capacity to an existing cluster, running node repair processes or replacing a dead node, and changing the attributes/settings of data containers.

MongoDB

By default, MongoDB supports only CLI tools, but external UI tools could be used. Compared to the tools that come with Cassandra or Couchbase, the standard MongoDB administration tools are rather immature.

Couchbase Server

Couchbase Server has a good Web administration UI. It provides all possible metrics per cluster, per node, and per bucket. All the maintenance tasks can be run through the Web UI, CLI, or REST API.

Cassandra (DataStax)

It is possible to perform all the maintenance tasks manually through the console using command line utilities that are bundled with Cassandra or through the DataStax OpsCenter.

Summary

Couchbase Server and Cassandra with OpsCenter provide similar functionality. However, Couchbase Web UI gives plenty of additional monitoring capabilities that are useful for correcting errors and optimizing resource utilization. Out-of-the-box MongoDB provides some CLI maintenance tools, but third-party Web UI solutions are also available.

Table 10. Maintenance of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
7	10	9

3.2.6. Recovery

This section evaluates policies and procedures to enable recovery of the databases after single-node failures.

MongoDB

Recovery is provided using the backup method. In cases where MongoDB works in a replica set, data can be recovered from another node of the set automatically.

Couchbase Server

Couchbase Server provides non-live recovery from backups and the CLI utility to restore data. It uses the swap rebalance functionality. When a node fails, you should either clean up and add a failed node again, or add a new node and perform a rebalance. The rebalance will be handled as a swap rebalance, which will minimize data movements and will not affect the overall capacity of the cluster.

Cassandra (DataStax)

When data becomes unsynchronized, the recovery process starts during regular database operation. Hinted handoff allows for full write availability. Data for reads can be repaired through read repair requests. If serious issues occur, data can be restored from a snapshot or from other replicas using Cassandra’s full-fledged utilities.

Summary

All of the three solutions have similar functionality for recovering failed nodes.

Table 11. Recovery of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	10

3.2.7. Monitoring functionality

This section gives an overview of the capabilities responsible for monitoring resources, performance, and statistics.

MongoDB

There is a set of utilities distributed with MongoDB for reporting database activities in real time. In addition, there are commands that return statistics on the current state of the database. There is the MMS Monitoring service, which collects data on the state of running MongoDB deployments and visualizes this data.

Couchbase Server

The administration UI provides a centralized view of all cluster metrics. The metrics are split into several major categories, such as server resources, vBucket resources, disk queues, TAP queues, XDCR operations, and a summary. One can drill down on the metrics to get an idea of how well a particular server is functioning or if there are any areas that need attention. The metrics are well-described in the Couchbase Server documentation.

Cassandra (DataStax)

To detect any performance issues, check out health characteristics, and plan scaling in/out a cluster, Cassandra exposes the most important metrics and operations over a standard protocol. It is possible to read and analyze these metrics with the nodetool command line utility, which operates over JMX.

A great alternative to that is DataStax OpsCenter, which is a graphical user interface for cluster monitoring and administration.

Summary

In this category, only Couchbase provides a centralized graphical UI with all cluster metrics. Cassandra has OpsCenter, but it is less comprehensive and only available in the DataStax Enterprise version. MongoDB uses a variety of utilities and tools for monitoring.

Table 12. The monitoring functionality of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
7	10	8

3.2.8. Security

This section reviews data storage and transfer security.

MongoDB

MongoDB authentication mechanisms include a password-based challenge, a response protocol, and x.509 certificates. Additionally, MongoDB Enterprise supports LDAP proxy authentication and Kerberos authentication for better data protection.

Couchbase Server

When configuring XDCR across multiple clusters over public networks, data is sent unencrypted across the public interface channel. To ensure security for the replicated information, you should configure a suitable VPN gateway between the two data centers. The cross-datacenter (XDCR) data security feature (Enterprise Edition only) provides secure cross data center replication using Secure Socket Layer (SSL) data encryption. Stored data can be encrypted using the Vormetric connector.

Notes from an architect's point of view:

Initially, the Memcached protocol was not designed to ensure security. It is highly recommended to design the access policy thoroughly, use SASL, and not to overlook firewalls.

Cassandra (DataStax)

Cassandra supports password-based authentication. The Kerberos authentication is not available in the open-source version, but is offered in the DataStax Enterprise version.

Authorization control allows for restricting the access rights of users on the table level. Traffic sent between the nodes and data centers can be encrypted, if the corresponding internode encryption property is enabled.

DataStax Enterprise allows for protecting data on disk that has been flushed from the Memtable in system memory to the SSTables.

Summary

Being much more mature, relational databases, generally have better support for security features—compared to NoSQL data stores.

MongoDB provides the best security among the evaluated solutions (with LDAP proxy, Kerberos authentication, x.509 certificates, etc.). Cassandra scores second with Kerberos authentication and encryption only available in the DataStax Enterprise edition. Couchbase may become vulnerable if you do not take all the recommended precautions.

Table 13. Security of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
7	5	6

3.3. Deployment

3.3.1. Availability

Availability is the ability to access a cluster, even if a node goes down.

MongoDB

A MongoDB cluster provides high availability using automatic failover. Failover enables a secondary member of the cluster to turn into a primary node, if the current primary node is unavailable. However, you will not be able to access the data on that node during the voting phase, which may last for a few seconds. If you enable reading from secondaries (eventual consistency), you can read data from them during the voting phase, although you cannot write new data. The failover mechanism does not require manual intervention. If you use a sharded cluster, Config servers should be located in at least two different power/network zones.

Couchbase Server

Couchbase server is a CP solution in terms of the CAP theorem. If a node with a primary replica fails, it will not be possible to perform write operations until the node is "failed over." You can enable automatic failover (which is disabled by default). Thirty seconds is the minimal delay for automatic node failover. Strong consistency at the document level is guaranteed—since all the operations on a particular key are served by a single node at a time while replication is asynchronous.

Notes from a developer's point of view:

The API is flexible, since the "replica read" feature allows for reading data before the failover has been finished. However, data may be inconsistent in this case.

Cassandra (DataStax)

Cassandra appreciates high availability and supports fault tolerance and high availability due to its design. Cassandra uses replication to achieve high availability, so that each row is replicated at N hosts, where N is the replication factor. Any read/write request for a key gets routed to any node in the Cassandra cluster; an application developer can specify a custom consistency level for both reads and writes on a per-operation basis.

For writes, the system routes requests to the replicas and waits for a specified quorum of replica nodes to acknowledge the completion of the writes. For reads—based on the consistency guarantees required by the client—the system either routes requests to the closest replica or routes requests to all replicas and then waits for a quorum of responses. Additionally for writes, the hinted handoff mechanism provides absolute write availability at the cost of consistency.

Notes from a developer's point of view:

High availability is interconnected with replication and consistency settings; therefore, it can be controlled in the application code. Replication guarantees that row copies are stored on replica nodes. The number of replicas is specified when a keyspace is created, so you can choose from the replica placement strategy options. The consistency level refers to how up-to-date and synchronized a row of data on all of its replica nodes is. For any given read or write operation, a client call specifies the level of consistency, which determines how many replica nodes must acknowledge the request. This feature is supported from the Thrift and CQL programming interfaces.

Notes from an architect's point of view:

Cassandra provides high availability out-of-the-box, since it is implied by its architecture. It works in a synchronous way with partitioning, replication, and failure handling features to handle read/write requests.

Summary

Due to automatic failover in MongoDB, data on a failed node becomes unavailable for a few seconds during the voting phase. Enabling eventual consistency makes it possible to read data from the replica set. In Couchbase, you cannot write to the failed node until it is failed over (with a minimal delay of 30 sec). Cassandra was designed for high availability and fault tolerance, so it is definitely the best data store in this category.

Table 14. Availability of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
8	9	10

3.3.2. Stability

This section evaluates the stability of APIs, interfaces, contracts, and also highlights known exploitation issues.

MongoDB

There is a long list of notable customers who are using MongoDB for their production deployments, including systems with terabytes of data and billions of documents. Still, we have detected an issue with the balancer in sharded setups. When the balancer moves chunks of data, but does not remove the chunk from its original shard, queries are broadcast to all shards.

Couchbase Server

Couchbase Server is based on two mature solutions: Memcached and CouchDB, which have been used for production deployments for 10+ years in top Web applications. However, when the metadata fills 100% of the available RAM size, Couchbase Server becomes unable to perform write operations.

Cassandra (DataStax)

With over six years since its first stable release, Cassandra 2.0 brings more stability and valuable functionality. See the full list of fixes and changes:

https://github.com/apache/cassandra/blob/trunk/CHANGES.txt

Summary

All the considered solutions are well-tried and market-proven to bring effectiveness and functionality with each regular release. Couchbase and MongoDB, however, have some minor issues. (Please note that while the score in this document reflects Couchbase Server v2.5, version 3.0 already supports tunable memory.)

Table 15. Stability of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
9	9	10

3.4. Development

3.4.1. Documentation

This section reviews user guides, white papers, online help, quick-reference guides, etc.

MongoDB

There is detailed official documentation from MongoDB, Inc.: http://docs.mongodb.org/manual/

Couchbase Server

All the versions of Couchbase Server are well-documented for administrators and developers.

Notes from a developer's point of view:

Most popular questions and possible issues are widely discussed in the Couchbase community.

Notes from an architect's point of view:

The Couchbase blog covers many architecture-related topics.

Cassandra (DataStax)

There is regularly updated documentation for developers and administrators on installation, configuration, and functionality of the open source version of Cassandra. In addition to that, there is a support forum on the DataStax Web site, as well as trainings offered by the company.

Notes from a developer's point of view:

Cassandra's developer forum and online documentation include examples and detailed guides on how to develop apps using the official DataStax C#, Java, or Python open-source drivers.

Summary

All the considered NoSQL solutions have a competitive level of documentation.

Table 16. Documentation of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	10

3.4.2. Integration

This section gives an overview of the ways and ease of linking together the common distributed computing solutions and software applications into one system.

MongoDB

MongoDB has a connector for Hadoop, which allows for using this system as an input source and/or an output destination. MongoDB runs well on Amazon EC2 and can be deployed from a pre-configured AMI.

Couchbase Server

There is the Sqoop plugin for Couchbase, which allows for streaming keys into HDFS or Hive to enable processing data with Hadoop. The elasticsearch plugin provides a full-text search through the documents. Just like MongoDB, Couchbase is also available as AMI and has some large customers running on Amazon EC2.

Cassandra (DataStax)

Hadoop can easily use data from Cassandra. The Hadoop support package features the same interface as HDFS to achieve data locality. DataStax Enterprise created a simplified way of integrating Cassandra into the product and also added a built-in analytics and enterprise search based on Solr.

Cassandra comes with EC2Snitch to enable deployment on the Amazon EC2 infrastructure. DataStax AMI facilitates installation by allowing users to set up a community cluster with the Management Console and to deploy a Cassandra cluster within a given availability zone.

Summary

All three data stores have Hadoop and Amazon EC2 infrastructure support.

Table 17. Integration capabilities of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	10

3.4.3. Support

This section reviews how the three enterprises provide assistance to users of their databases.

MongoDB

MongoDB offers commercial and community support. There are three subscription levels—*enterprise*, *standard*, and *basic*—that provide customers with enterprise-grade functionality, uptime, and support. There is a reasonably large developer community (and it's growing) and a user group where you can get help and exchange ideas with others.

Couchbase Server

Commercial and community support, as well as various webinars and trainings, are available on the Couchbase Web site.

Cassandra (DataStax)

Enterprise and community support.

Summary

All of the three considered solutions are widely-used and have commercial support.

Table 18. Enterprise and community support of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	10

3.4.4. Usability

Usability of a system is the ease of effective use and learnability.

MongoDB

It is relatively easy to deploy MongoDB when you use a replica set and relatively complicated when you need a sharded cluster. Out of the box, MongoDB does not include a GUI. Instead, most administration tasks are completed from the command line tools, such as Mongo shell. However, some UI solutions, such as MongoDB Management Service (MMS), are available as an external service.

Couchbase Server

Couchbase Server is simple, fast, and elastic. Its Web UI allows for monitoring and administering the cluster, as well as for creating and testing indexes and queries.

Notes from a developer's point of view:

It has a flexible document-based data model. Views implement MapReduce with JavaScript. They are utilized for indexing and JSON querying. In addition, there is Couchbase Query Language (N1QL), which is similar to CQL in Cassandra.

Notes from an architect's point of view:

The system has schema-less data modeling.

Cassandra (DataStax)

It is quite easy to deploy, configure, and use a Cassandra cluster. Such tools as OpsCenter and DevCenter simplify its administration. OpsCenter is a Web interface, which runs a centralized management console for standard installation and maintenance operations. DevCenter is a free, Eclipse-based tool, which is a visual interface for creating and running CQL queries and commands that can be used for open-source and DataStax distributions of Cassandra.

Summary

Couchbase Server is a bit easier to understand and user-friendlier than Cassandra. However, Cassandra's CQL is usually more familiar to developers than the Views concept provided by Couchbase. MongoDB is very convenient for dynamic queries, but too complicated in terms of deployment and general architecture ideas.

Table 19. Usability of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
8	9	9

3.5. Performance and scalability

3.5.1. Performance

System performance is responsiveness and stability under a particular workload. Workload is described in terms of read/insert/update/delete proportion and issued throughput in operations per second. Responsiveness is measured as latency in milliseconds (ms).

MongoDB

MongoDB may experience performance issues during writes, because it uses a reader-writer block that allows for concurrent reads, but locks write operations. The main requirement for improving the performance of MongoDB is using data sets that fit memory.

Performance tests for MongoDB were carried out on a replica set that consisted of three members: a master node and two slaves. One separate node was used as a client.

By default, read and write operations of a replica set in MongoDB are sent to the primary node. So, the results are consistent with the last operation. However, in this case, a primary node becomes a bottleneck.

There are several options for improving performance. For example, a client may be configured with a read preference, which means that read operations will be firstly directed to the secondary members. It will dramatically improve read performance. Keep in mind that if you use this setting and allow clients to read secondary reads, reads can be returned from secondary members that have not been replicated yet and—therefore—do not contain the most recent updates. This can be caused by a lag between master and slave nodes.

To guarantee consistency for reads from secondary members, you can specify the setting that would recognize a write operation’s completeness only if it has been succeeded on all of the nodes. In this case, you will achieve full consistency, but it will lead to slowing down write operations. A test client we used wrote data only to the primary node and read data from all replicas. It helped us to increase read/write performance, but this corresponds to the eventual consistency level.

One of the main requirements for achieving good MongoDB performance is using working data sets that fit memory. To map data files to memory, MongoDB uses a memory-mapped file operation system mechanism. By using memory-mapped files, MongoDB can deal with the contents of its data files as if they were in memory.

MongoDB stores its data in the files called extents with the standard size of 2 GB; however, the process is actually more complex. Such files are created on demand as the database grows. To increase efficiency and reduce disk fragmentation, the whole file is pre-allocated. The data files include files that contain no data, but the space for them has been allocated. For example, MongoDB can allocate 1 GB to a data file, which may be 90% empty.

Actually, stored data has significant overhead compared to the size reserved for documents. This explains why it is required to allocate more RAM than the actual size of a working set.

One more drawback of MongoDB—that affects performance of write operations—is the reader-writer lock on the database level. The lock allows concurrent reads to access a database, but gives exclusive access to a single write operation, which significantly reduces the performance of the solution under write-intensive workloads.

Couchbase Server

A managed object cache based on Memcached provides predictable latency of a sub-millisecond for read and write operations. CRUD operations update RAM cache and are then inserted into the persistence queue for optimizing disk I/O. It makes Couchbase Server eventually persistent. Asynchronous communication is supported.

Couchbase Server was tested with a synchronous API using the "replicate to zero" and "persist to zero" options. That means the CRUD operations updated RAM cache and returned results immediately, while replication and persistence were asynchronous. You can enable/disable these options by calling particular parameters.

Notes from a developer's point of view:

Asynchronous and synchronous APIs are available. It is possible to balance between performance and persistence using an API of the SDK for each particular CRUD operation.

Cassandra (DataStax)

Cassandra is optimized for intensive writes. Key and row caching can greatly accelerate reads when there are a large number of rows accessed frequently.

Cassandra behaves as a very effective solution for write-prevailing loads; it delivered an average performance of less than 1 ms for inserts, updates, and deletes—accompanying it with predictable, stably growing throughput. Fast updates are empowered by its architecture, where updated data is simultaneously written to an in-memory structure called Memtable and saved to the transaction commit log on a disk for persistency.

Cassandra can also be very responsive under read-intensive workloads. Indeed, it showed a quite decent throughput of 25 K ops/sec with maximum latency time squeezing into 3–4 ms intervals. Reads are highly dependent on JVM and garbage collection settings used on the cluster's nodes, since garbage collection activities stop the application while it frees up memory and make nodes become unresponsive during the running time. The column family used for benchmarking was created with settings that enabled both key cache and off heap cache for row data to lower on-disk pressure and speed up read latencies. Additionally, we increased the new generation size from its default settings. The bigger the younger generation, the less often minor collections occur and less often the client experiences garbage collection pauses.

Summary

For the performance tests, we used one YCSB client and three Couchbase nodes. From the charts, you can see that Couchbase mainly outperforms MongoDB and Cassandra in the maximum throughput when data fits into memory (see the tests with 30 million records, 10 KB each). However, performance of this solution tends to be limited by the network bandwidth of 1 GB, while other databases do not have such a boundary. When data do not fit memory (approximately 60% in RAM, 40% on disk), performance of Couchbase decreases, but it closely follows the results demonstrated by Cassandra.

Table 20. Performance of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
6	10	8

3.5.2. Scalability

Scalability is the ability of a solution to handle a growing amount of data and cluster loads. Here, we are considering only horizontal scaling (e.g., adding more nodes to the system).

MongoDB

MongoDB has two scalability options:

a) Focus on read operations by *master-slave* replication.
b) Focus on read and write operations by *sharding*.

The first one is a replica set, a group of mongod processes that maintain the same data set and support master-slave replication. Depending on the settings, a replica set allows for reading from replicas, but always sends write operations to the master node. By adding more nodes to a replica set, you can achieve almost linear scalability of read operations, which was proved by our research and tests. The process of data migration from a master node to the added replica nodes runs automatically. The process of adding new nodes to a replica set is quite simple and can be done by running a single command. So, a replica set is the crucial concept for all production deployments.

In cases when performance of a replica set is limited by system resources, the performance of the system may go down dramatically. Examples of such cases:

- High query rates can exhaust the CPU capacity of the server.
- Large data sets may exceed the storage capacity of a single machine.
- The size of a working data set that is larger than the system's RAM imposes an extra load on the I/O capacity of disk drives.

One more case of performance degradation is when the system works under a write intensive workload. MongoDB uses a reader-writer lock on a per-database basis. It allows for concurrent reads, but gives exclusive access to a single write operation. So the reader-writer lock reduces concurrency of write operations.

The second one is sharding, the method for storing data across multiple machines or a set of machines. MongoDB uses sharding for deployments with very large data sets and high write operation throughput. One needs to deploy an additional replica set and bind it with the existing one to set up a sharding cluster. Data is split and migrated to the new replica set automatically by an assigned key. Additionally, a sharded cluster requires deploying three config servers and several mongos processes. Config servers store cluster metadata information, and mongos processes split requests between shards using a predefined key.

To summarize, deploying a sharded cluster is rather complicated. You should know the data model and access patterns, as well as run additional commands to make all components of the deployment work together.

Couchbase Server

When the number of servers in the Couchbase cluster changes because of scaling or failures, data partitions must be redistributed. This ensures that data is evenly distributed throughout the cluster and that application access to the data is load-balanced across all the servers.

One can trigger rebalancing using an explicit action from the admin UI or through a REST call. When a rebalance begins, the rebalance orchestrator calculates a new cluster map based on the current pending set of servers to be added and removed from the cluster. It streams the new cluster map to all the servers in the cluster. During rebalance, the cluster moves data via partition migration directly between two server nodes in the cluster. As the cluster moves each partition from one location to another, an atomic and consistent switchover takes place between the two nodes, and the cluster updates each connected client library with an actual cluster map.

Notes from an architect's point of view:

The system scales up and down easily, with three steps:

- Single package installation
- Adding a node to the existing cluster
- Starting the rebalancing process

It is possible to start with a POC for a single node and later grow it into a clustered solution. In this case, software solution redesign is not required.

You may also clone any Couchbase node in a virtualized environment and add the node to the cluster. However, this feature may look redundant, since Couchbase Server is installed as just a single package on most of the platforms.

The line chart **(Chart 1 – Throughput in Appendix C)** demonstrates how performance of the client and server (in operations per second) changed when the number of nodes increased. In all the tests, we used six clients, while the number of nodes grew from three to six. Each of the clients ran 100 threads with read, insert, update, and delete queries in proportions specified above the workload.

We used the 1,000 Mb/s network connection, so the maximum expected insert (read) throughput was 1,000 (Mb/s) * 1,048,576 (bits in megabit) / 8 (bits in byte) * 1,024 (a 1K record) = 128,000 ops/sec. In reality, we achieved the maximum average throughput of ~111,000 ops/sec when the system had six clients and four server nodes. The actual average throughput in all the tests was 104,000 ops/sec, which is ~86% network utilization. CPU, RAM, and disk were underused during these tests.

Other charts show dependency between the latency and the number of nodes. For Couchbase, the asymptotic best average latency was ~1 ms.

Cassandra (DataStax)

Cassandra can provide almost linear scalability. Adding a new node or removing an old node from a cluster requires performing some operations. They can be implemented from the nodetool command line helper or through the DataStax OpsCenter.

Cassandra allows for adding new nodes dynamically, as well as for adding a new data center to the existing cluster. With Cassandra under v1.2, scaling out an existing cluster required a more thorough understanding of the database architecture and included some manual steps. Such deployments had one token per node, so a node owned exactly one contiguous range in the ring space. When a new node was added to a cluster, you were to calculate a new token for a node, re-calculate tokens for the cluster manually, then assign new tokens to the existing nodes with nodetool, and eventually remove unused keys on all nodes using nodetool cleanup. Besides that, the initial token property could be left empty. As a result, the token range of a node that was working under the heaviest load would be split and a new node would be added.

The paradigm described above was changed with the release of Cassandra v1.2, which has virtual nodes or *vnodes*, making the legacy manual operations unnecessary. Unlike the previous versions that had one token or a range of tokens per node, Cassandra v1.2 has many tokens per node. Within a cluster, vnodes can be selected randomly and be non-contiguous.

Vnodes greatly simplified scaling out an existing cluster. You do not have to calculate tokens, assign them to each of the nodes, and rebalance a cluster. In the updated version of Cassandra, a new node gets an even portion of the data. To add a new node, the existing cluster should be introduced (you should set a few connections and auto-bootstrap properties). After that, a Cassandra daemon will start on each of the new nodes. As a final step, you can call nodetool cleanup during low-usage hours to remove keys that are no longer in use.

Adding a new node from the OpsCenter GUI is even simpler. You should click Add Node in the cluster view and provide sudo credentials for authentication.

Reducing the cluster size is also straightforward. You can do it from the command line of the nodetool utility. Firstly, you should run a drain command to stop accepting writes from the client and flush memtables on a particular remaining node. Secondly, run a decommission command to move data from the removed nodes to other nodes. Finally, complete removal of the node with the nodetool removenode operation.

The scalability tests showed that Cassandra scales out linearly by adding more computing resources. It performed 46,000 ops/sec on three nodes, up to 56,000 ops/sec on four nodes, up to 61,000 ops/sec on five nodes, and up to 71,000 ops/sec on six nodes (see the charts in Appendix C).

Summary

Couchbase Server and Cassandra are easy to scale. The scaling process is predictable and does not require additional cluster design actions as in the case of MongoDB.

Table 21. Scalability of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
8	10	10

4. Summary (a comparative table)

The table below summarizes the points scored by MongoDB, Cassandra, and Couchbase Server for each criterion (on a scale of 1–10). Here, we assumed that all criteria are equal in terms of importance.

However, one can use this scorecard to select an appropriate NoSQL solution for a particular use case. To do this, choose a weight for each of the criteria according to your project needs. After that, multiply basic scores by the weights and calculate total weighted scores to make a final decision.

Table 22. Comparing MongoDB, Couchbase Server, and Cassandra

Criteria		MongoDB		Couchbase Server		Cassandra (DataStax)	
Definition	Weight	Basic Score	Weighted Score	Basic Score	Weighted Score	Basic Score	Weighted Score
1. Installation and configuration		6		10		9	
2. Topology		7		10		9	
3. Consistency		10		10		9	
4. Fault tolerance		7		10		10	
5. Structure and format		10		10		9	
6. Audit and control		7		9		9	
7. Configuration management		7		10		9	
8. Backup		9		9		10	
9. Disaster recovery		8		9		10	
10. Maintenance		7		10		9	
11. Recovery		10		10		10	
12. Monitoring		7		10		8	
13. Security		7		5		6	
14. Availability		8		9		10	
15. Stability		9		9		10	
16. Documentation		10		10		10	
17. Integration		10		10		10	
18. Support		10		10		10	
19. Usability		8		9		9	
20. Performance		6		10		8	
21. Scalability		8		10		10	
Total Scores:		**Basic**	**Weighted**	**Basic**	**Weighted**	**Basic**	**Weighted**
		171		**199**		**194**	

For more details on the report, feel free to contact us at engineering@altoros.com.

5. Appendix A: Configuring a benchmark environment

Tests were run on the dedicated servers of SoftLayer to provide precise and replicable results. The initial tests demonstrated that the selected NoSQL solutions performed much better on SoftLayer's bare metal compared to the Amazon virtual machines.

Configuration of a database node:

Component	Configuration
Processor	2 GHz Intel Xeon-SandyBridge (E5-2620-HexCore)
RAM	4x16 GB Kingston 16 GB DDR3 2Rx4
Hard Drive	960 GB SanDisk CloudSpeed 1000 SSD
OS	CentOS 6.5-64

Configuration of the YCSB client node:

Component	Configuration
Processor	3.5 GHz Intel Xeon-IvyBridge (E3-1270-V2-Quadcore)
RAM	2x4 GB Kingston 4 GB DDR3 2Rx8
Hard Drive	500 GB Western Digital WD Caviar RE4
OS	CentOS 6.5-64

6. Appendix B: Performance results

The following are the configuration parameters used on each of the YCSB client nodes to benchmark a database.

50 million 1 KB records:

```
fieldcount=10
fieldlength=100
threadcount=50
recordcount=50000000
```

3 million 10 KB records:

```
fieldcount=10
fieldlength=1000
threadcount=50
recordcount=3000000
```

The cluster was loaded using one client node. The benchmarking was performed on two data sets: the first is 50 million records (1 KB each), the second is 3 million records (10 KB each):

- Load: 1 KB, 50 mln (the data set did not fit the memory)
- Load: 10 KB, 3 mln (fit the memory)

6.1. Workload B

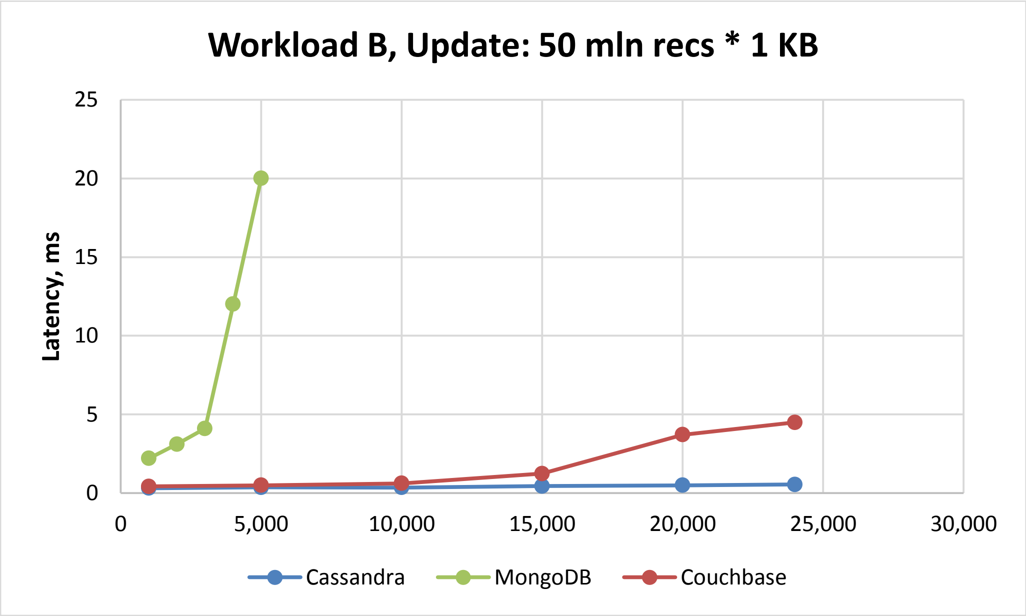
Workload B consisted of 50% read operations, 40% update operations, 5% insert operations, and 5% delete operations.

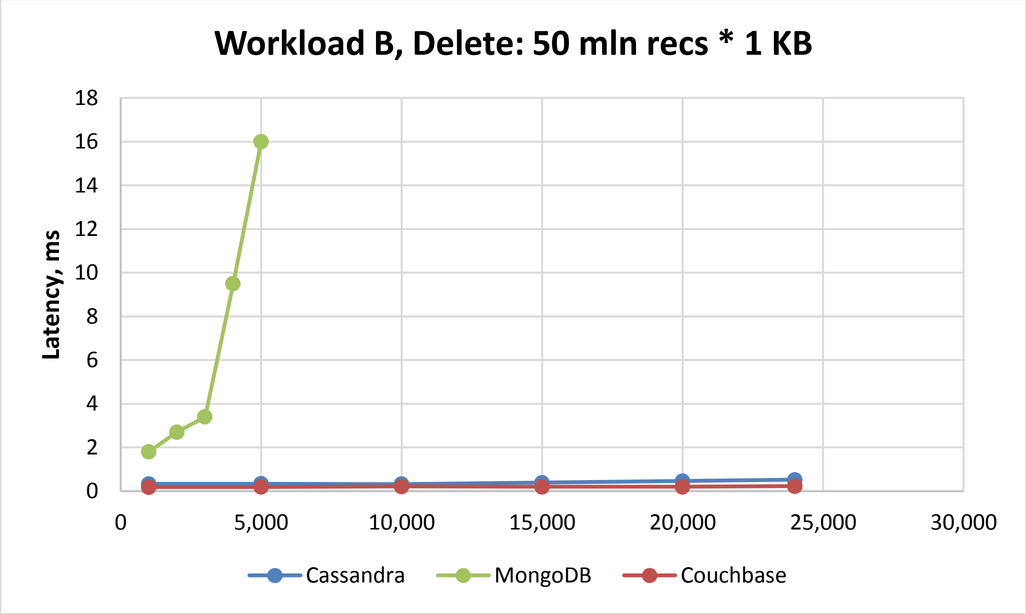
Configuration parameters for YCSB:

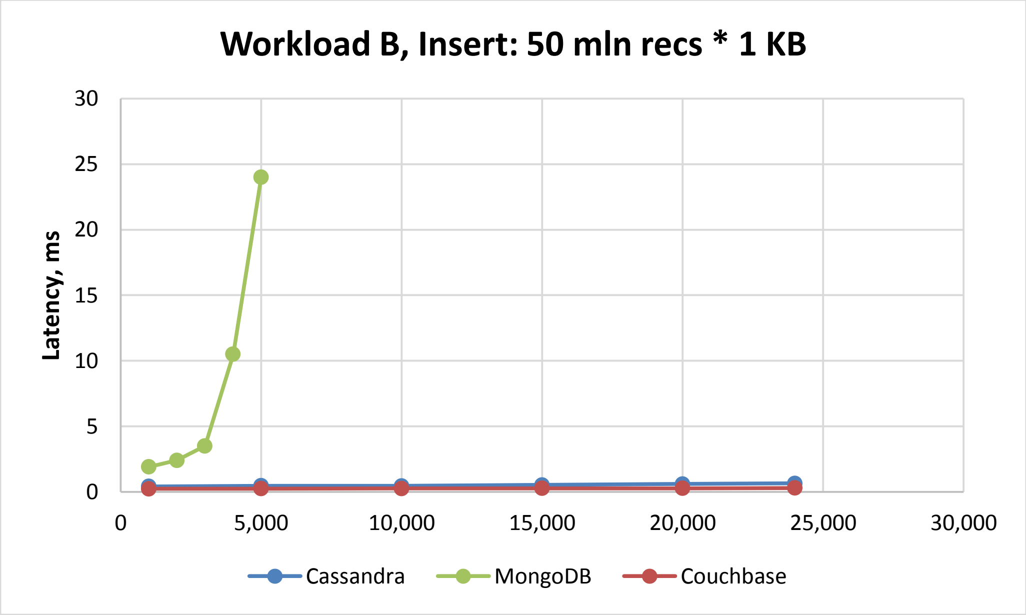
```
readallfields=true
readproportion=0.5
updateproportion=0.4
scanproportion=0
insertproportion=0.05
deleteproportion=0.05
requestdistribution=zipfian
```

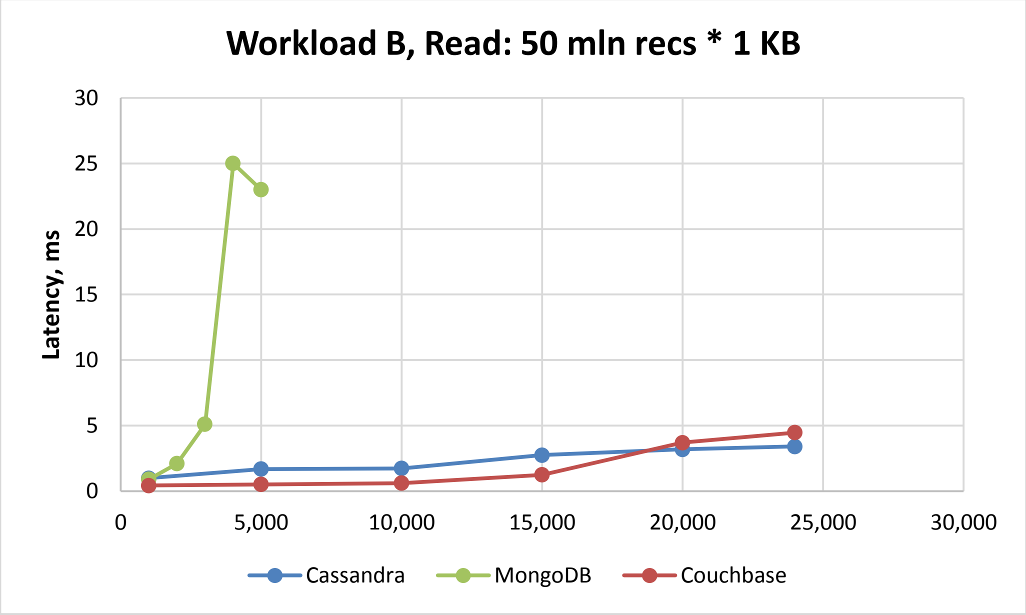
a) Workload B, 50 million records

	Cassandra (DataStax)					MongoDB					Couchbase				
50 mln recs, 1 KB each	29 K,1.7 ms					5.4 K,18 ms					87 K, 1.09 ms				
	Op/s	Upd	Del	Ins	Read	Op/s	Upd	Del	Ins	Read	Op/s	Upd	Del	Ins	Read
Workload B	1K	0.3	0.33	0.41	0.99	1K	2.2	1.8	1.9	0.9	1K	0.42	0.18	0.23	0.43
	2K					2K	3.1	2.7	2.4	2.1					
	3K					3K	4.1	3.4	3.5	5.1					
	4K					4K	12	9.5	10.5	25					
	5K	0.36	0.34	0.46	1.69	5K	20	16	24	23	5K	0.49	0.19	0.24	0.51
	10K	0.35	0.32	0.45	1.73	10K					10K	0.61	0.21	0.26	0.61
	15K	0.44	0.39	0.52	2.75	15K					15K	1.23	0.2	0.27	1.24
	20K	0.49	0.46	0.6	3.19	20K					20K	3.7	0.2	0.27	3.7
Workload Bmax	**24K**	**0.54**	**0.52**	**0.65**	**3.4**	**24K**					**24K**	**4.49**	**0.23**	**0.29**	**4.47**




Workload B, Delete: 50 mln recs * 1 KB
Latency, ms
0
2
4
6
8
10
12
14
16
18
0
5,000
10,000
15,000
20,000
25,000
30,000
Cassandra
MongoDB
Couchbase

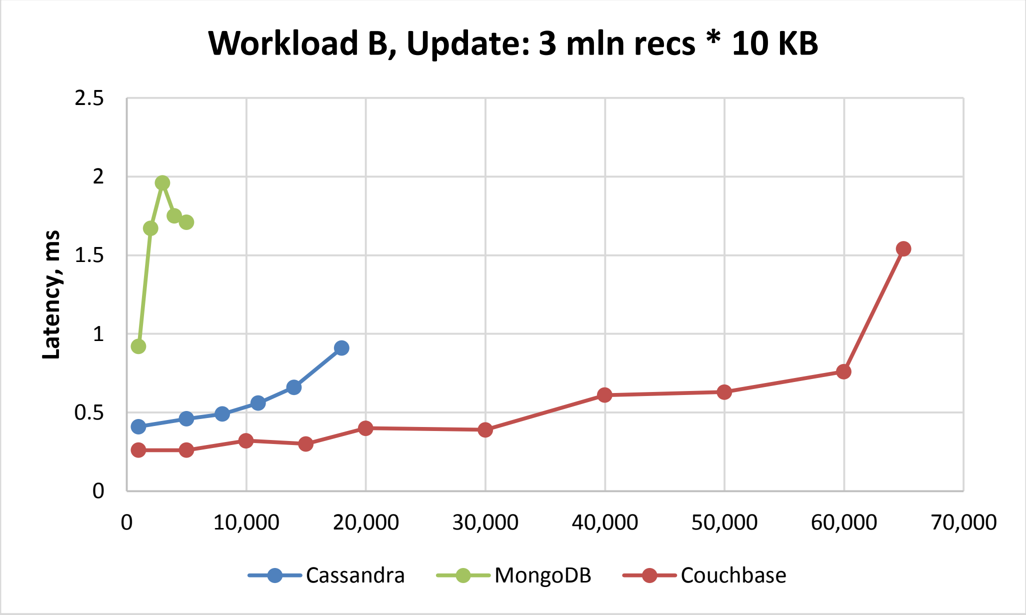

Workload B, Insert: 50 mln recs * 1 KB
Latency, ms
0
5
10
15
20
25
30
0
5,000
10,000
15,000
20,000
25,000
30,000
Cassandra
MongoDB
Couchbase

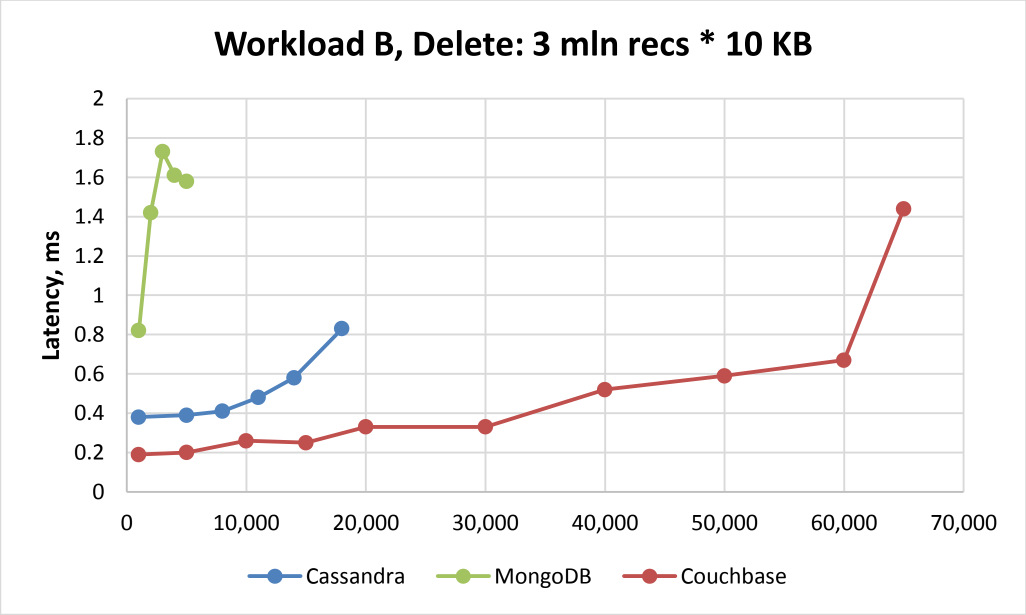


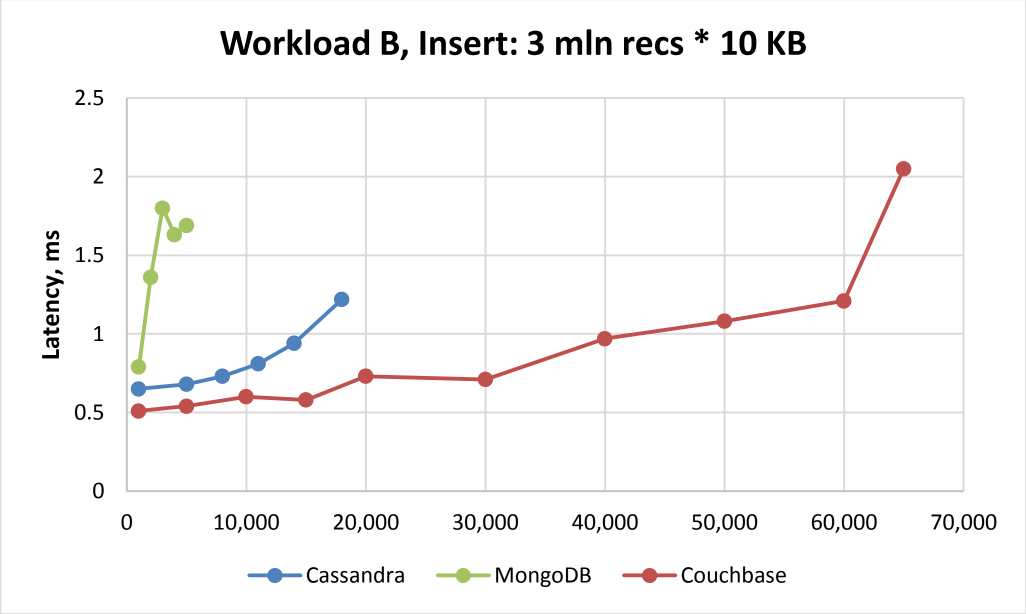
b) Workload B, 3 million records

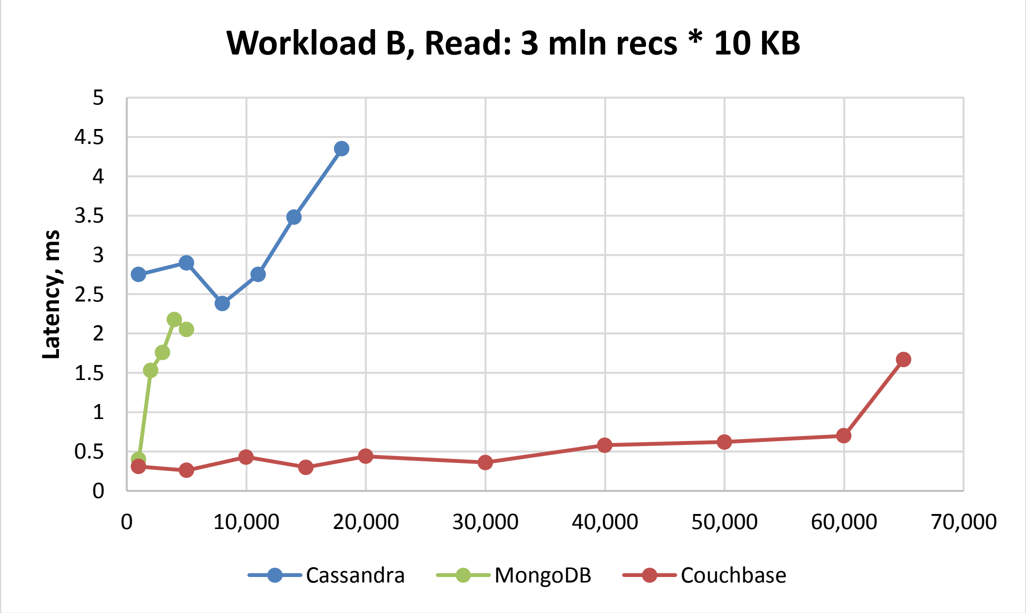
The total data set included 3 million records, 10 KB each.

	Cassandra (DataStax)					MongoDB					Couchbase				
3 mln recs, 10 KB each	7.5 K, 5.8 ms					2 K, 5.09 ms					10 K, 9.4 ms				
	Op/s	Upd	Del	Ins	Read	Op/s	Upd	Del	Ins	Read	Op/s	Upd	Del	Ins	Read
Workload B	1K	0.41	0.38	0.65	2.75	1K	0.92	0.82	0.79	0.4	1K	0.26	0.19	0.51	0.31
	2K					2K	1.67	1.42	1.36	1.53	2K				
	3K					3K	1.96	1.73	1.8	1.76	3K				
	4K					4K	1.75	1.61	1.63	2.18	4K				
	5K	0.46	0.39	0.68	2.9	5K	1.71	1.58	1.69	2.05	5K	0.26	0.2	0.54	0.26
	8K	0.49	0.41	0.73	2.38	8K					8K				
	10K					10K					10K	0.32	0.26	0.6	0.43
	11K	0.56	0.48	0.81	2.75	11K					11K				
	14K	0.66	0.58	0.94	3.48	14K					14K				
	15K					15K					15K	0.3	0.25	0.58	0.3
	18K	0.91	0.83	1.22	4.35	18K					18K				
	20K					20K					20K	0.4	0.33	0.73	0.44
	30K					30K					30K	0.39	0.33	0.71	0.36
	40K					40K					40K	0.61	0.52	0.97	0.58
	50K					50K					50K	0.63	0.59	1.08	0.62
	60K					60K					60K	0.76	0.67	1.21	0.7
Workload Bmax											65K	1.54	1.44	2.05	1.67


Workload B, Update: 3 mln recs * 10 KB
Latency, ms
2.5
2
1.5
1
0.5
0
0
10,000
20,000
30,000
40,000
50,000
60,000
70,000
Cassandra
MongoDB
Couchbase


Workload B, Delete: 3 mln recs * 10 KB
Latency, ms
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
10,000
20,000
30,000
40,000
50,000
60,000
70,000
Cassandra
MongoDB
Couchbase





6.2. Workload C

Workload C consisted of 90% read operations, 8% update operations, 1% insert operations, and 1% delete operations.

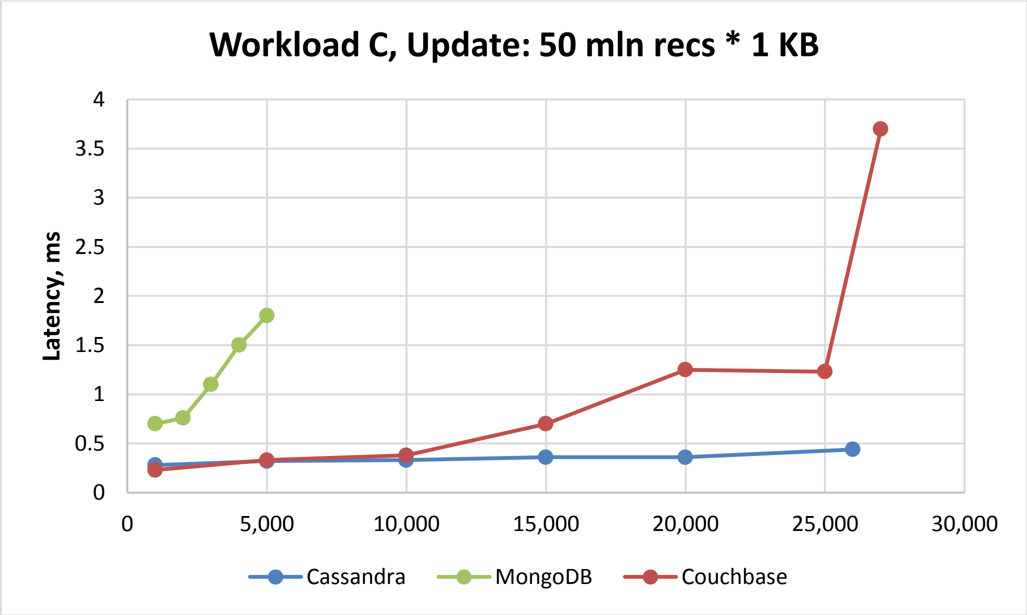
Configuration parameters for YCSB:

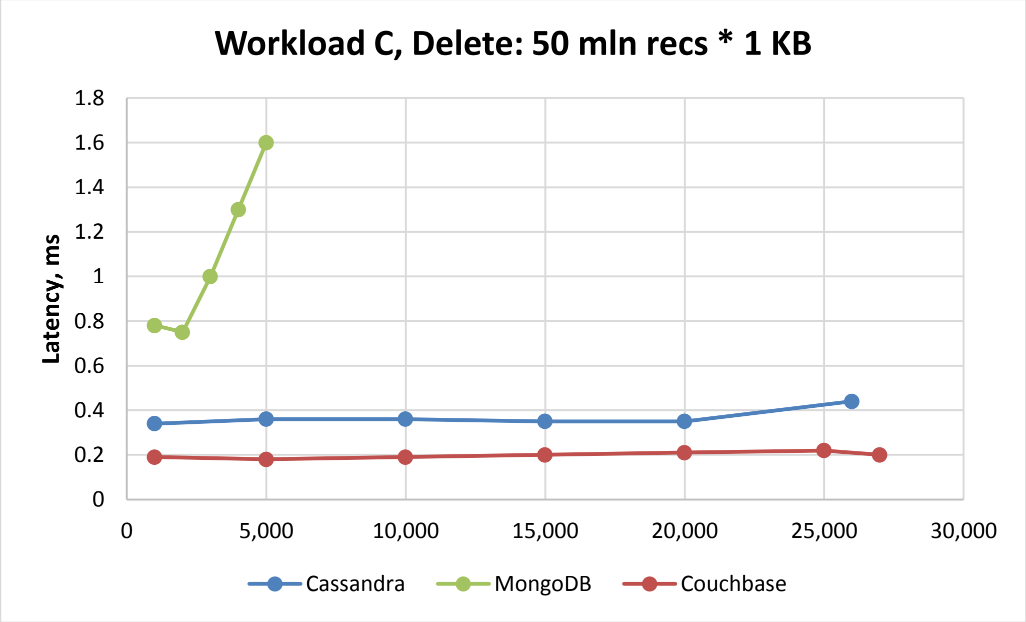
```
readallfields=true
readproportion=0.9
```

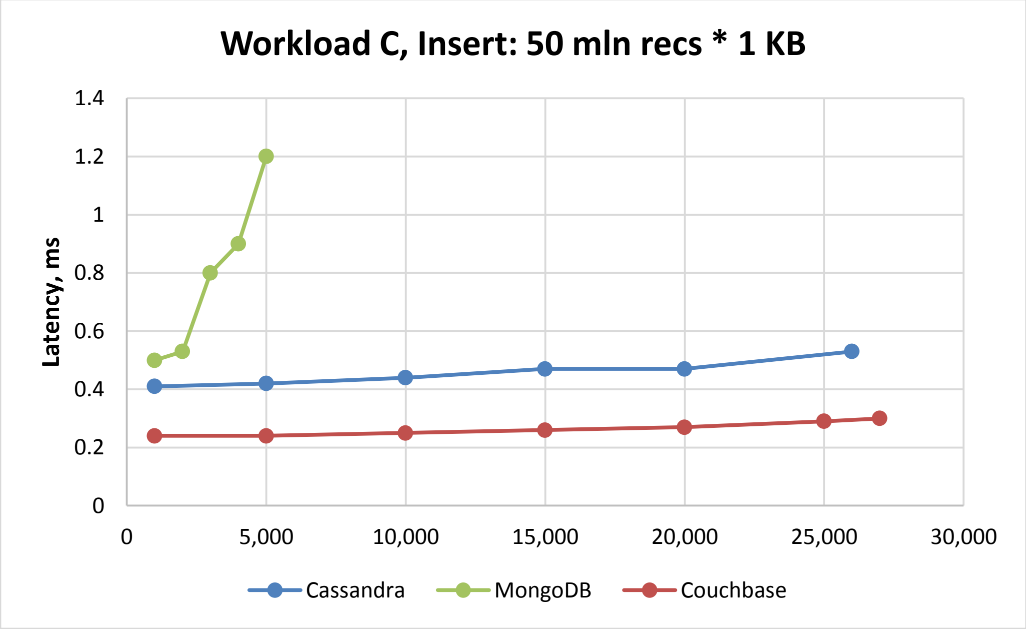
updateproportion=0.08
scanproportion=0
insertproportion=0.01
deleteproportion=0.01
requestdistribution=zipfian

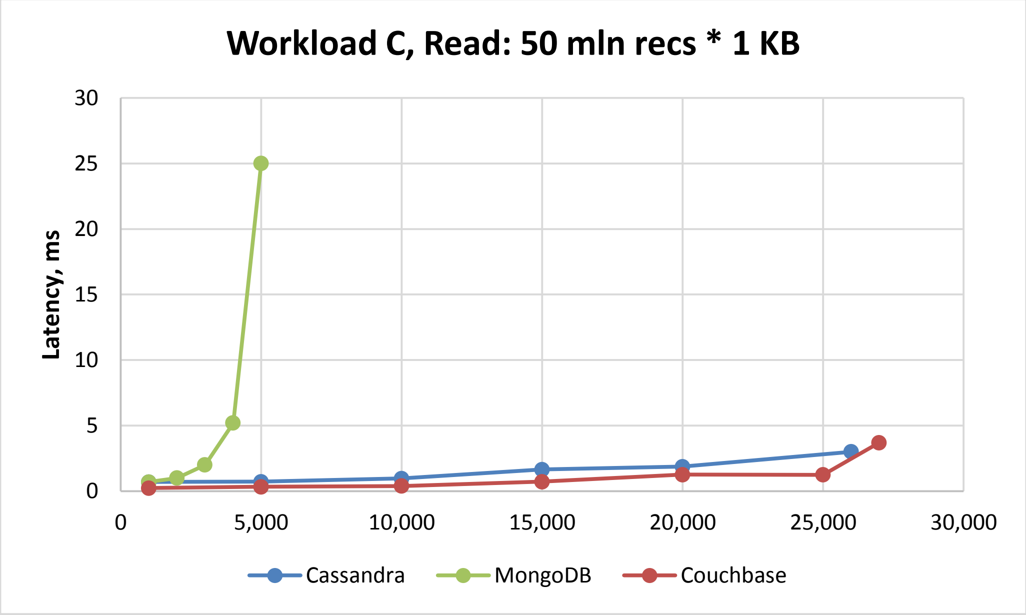
a) Workload C, 50 million records

	Cassandra (DataStax)					MongoDB					Couchbase				
50 mln recs, 1 KB each	29 K, 1.7 ms					5.4 K, 18 ms					87 K, 1.09 ms				
	Op/s	Upd	Del	Ins	Read	Op/s	Upd	Del	Ins	Read	Op/s	Upd	Del	Ins	Read
Workload C	1K	0.28	0.34	0.41	0.69	1K	0.7	0.78	0.5	0.7	1K	0.23	0.19	0.24	0.23
	2K					2K	0.76	0.75	0.53	1	2K				
	3K					3K	1.1	1	0.8	2	3K				
	4K					4K	1.5	1.3	0.9	5.2	4K				
	5K	0.32	0.36	0.42	0.72	5K	1.8	1.6	1.2	25	5K	0.33	0.18	0.24	0.33
	10K	0.33	0.36	0.44	0.96	10K					10K	0.38	0.19	0.25	0.39
	15K	0.36	0.35	0.47	1.65	15K					15K	0.7	0.2	0.26	0.72
	20K	0.36	0.35	0.47	1.87	20K					20K	1.25	0.21	0.27	1.26
	25K					25K					25K	1.23	0.22	0.29	1.24
	26K	0.44	0.44	0.53	3.001	26K					26K				
Workload Cmax	27K					27K					27K	3.7	0.2	0.3	3.7



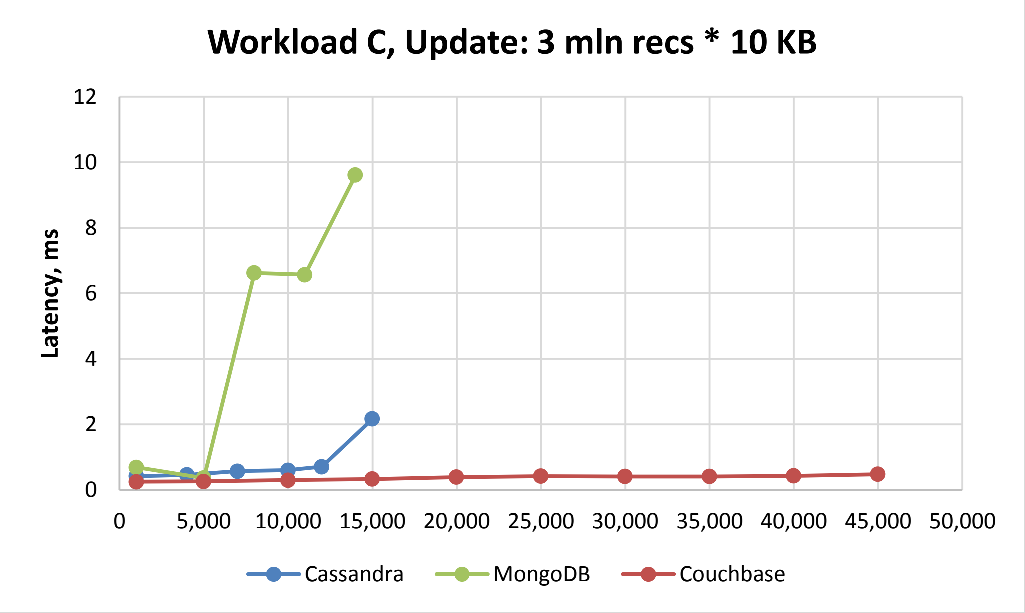

Workload C, Delete: 50 mln recs * 1 KB
Latency, ms
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
5,000
10,000
15,000
20,000
25,000
30,000
Cassandra
MongoDB
Couchbase

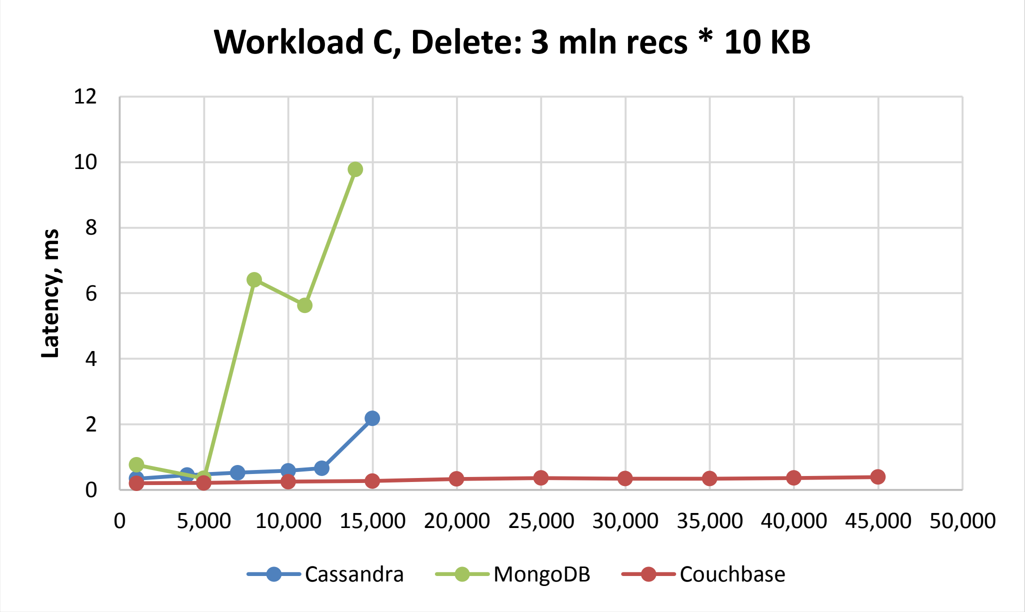

Workload C, Insert: 50 mln recs * 1 KB
Latency, ms
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
5,000
10,000
15,000
20,000
25,000
30,000
Cassandra
MongoDB
Couchbase

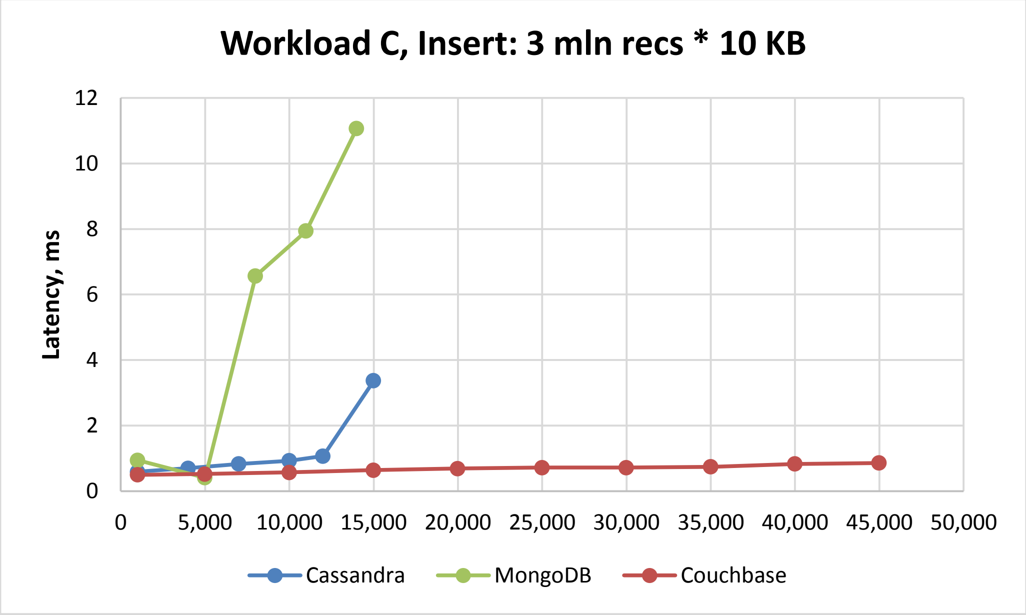


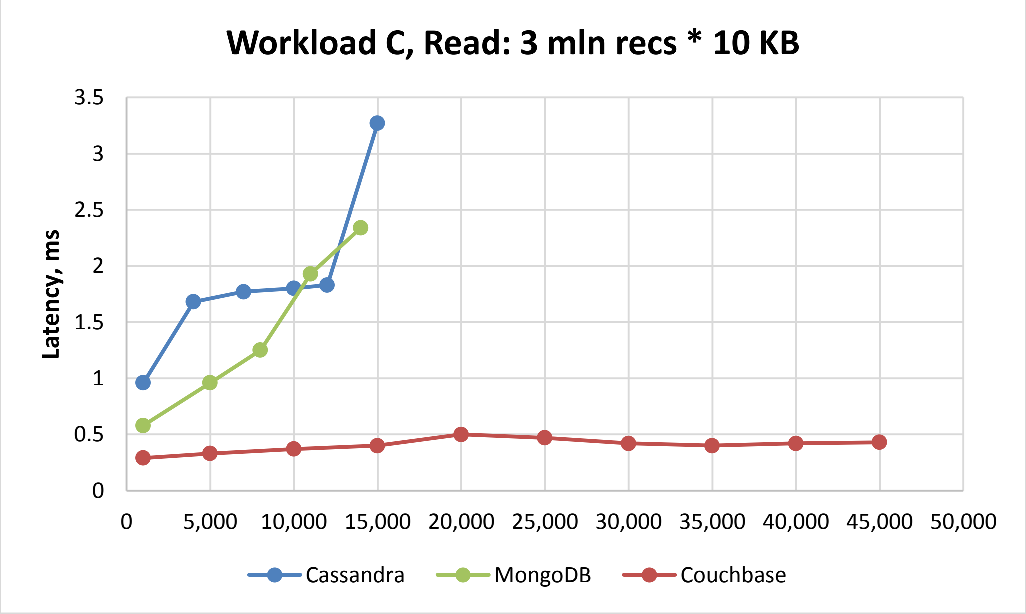
b) Workload C, 3 million records

	Cassandra (DataStax)					MongoDB					Couchbase				
3 mln recs, 10 KB each	**7.5 K, 5.8 ms**					**2 K, 5.09 ms**					**10 K, 9.4 ms**				
	Op/s	**Upd**	**Del**	**Ins**	**Read**	**Op/s**	**Upd**	**Del**	**Ins**	**Read**	**Op/s**	**Upd**	**Del**	**Ins**	**Read**
Workload C	**1K**	0.42	0.34	0.59	0.96	**1K**	0.69	0.76	0.94	0.58	**1K**	0.25	0.2	0.5	0.29
	4K	0.46	0.45	0.7	1.68	**4K**					**4K**				
	5K					**5K**	0.36	0.35	0.41	0.96	**5K**	0.26	0.21	0.52	0.33
	7K	0.57	0.52	0.83	1.77	**7K**					**7K**				
	8K					**8K**	6.62	6.41	6.57	1.25	**8K**				
	10K	0.6	0.58	0.93	1.8						**10K**	0.3	0.25	0.57	0.37
	11K					**11K**	6.57	5.63	7.94	1.93	**11K**				
	12K	0.71	0.66	1.07	1.83	**12K**					**12K**				
	14K					**14K**	9.61	9.78	11.07	2.34	**14K**				
	15K	**2.17**	**2.18**	**3.37**	**3.27**						**15K**	0.33	0.27	0.64	0.4
											20K	0.39	0.33	0.69	0.5
											25K	0.42	0.36	0.72	0.47
											30K	0.41	0.34	0.72	0.42
											35K	0.41	0.34	0.74	0.4
											40K	0.43	0.36	0.83	0.42
											45K	0.48	0.39	0.86	0.43
Workload Cmax															


Workload C, Update: 3 mln recs * 10 KB
Latency, ms
0
2
4
6
8
10
12
0
5,000
10,000
15,000
20,000
25,000
30,000
35,000
40,000
45,000
50,000
Cassandra
MongoDB
Couchbase


Workload C, Delete: 3 mln recs * 10 KB
Latency, ms
0
2
4
6
8
10
12
0
5,000
10,000
15,000
20,000
25,000
30,000
35,000
40,000
45,000
50,000
Cassandra
MongoDB
Couchbase





6.3. Workload D

Workload D consisted of 10% read operations, 72% update operations, 9% insert operations, and 9% delete operations.

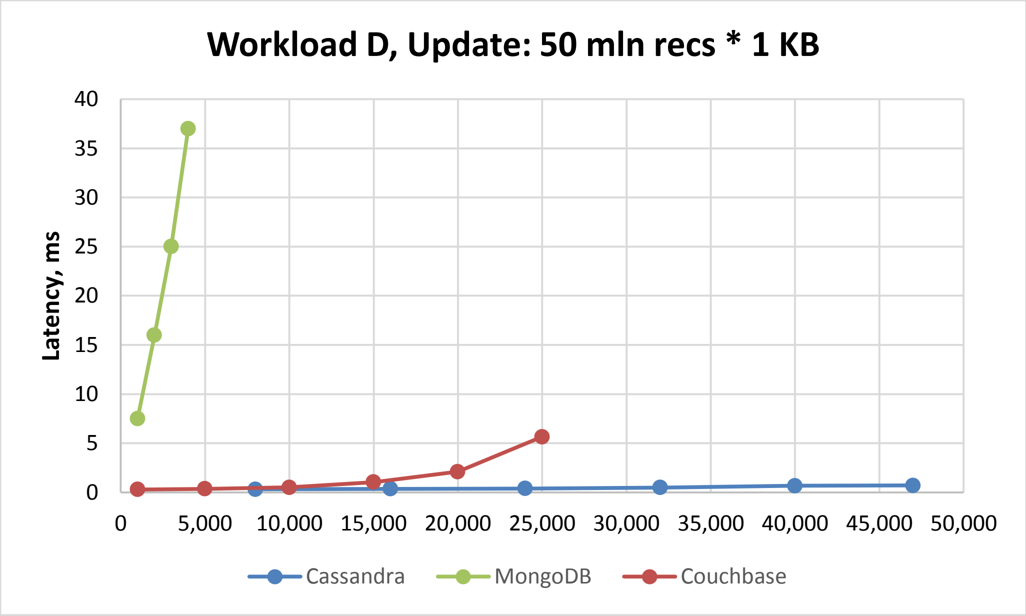
Configuration parameters for YCSB:

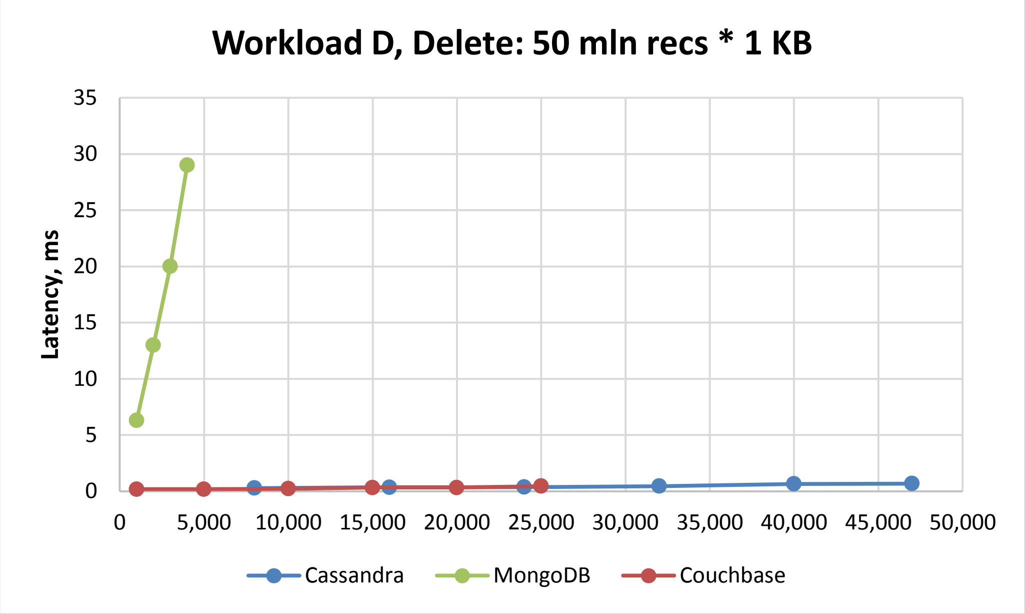
readallfields=true

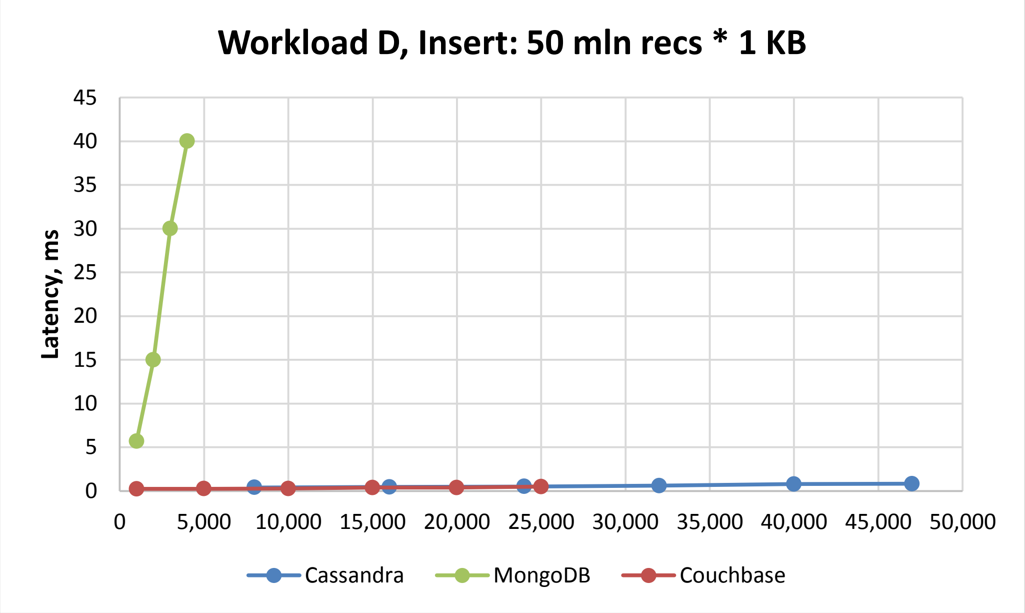
readproportion=0.1
updateproportion=0.72
scanproportion=0
insertproportion=0.09
deleteproportion=0.09
requestdistribution=latest

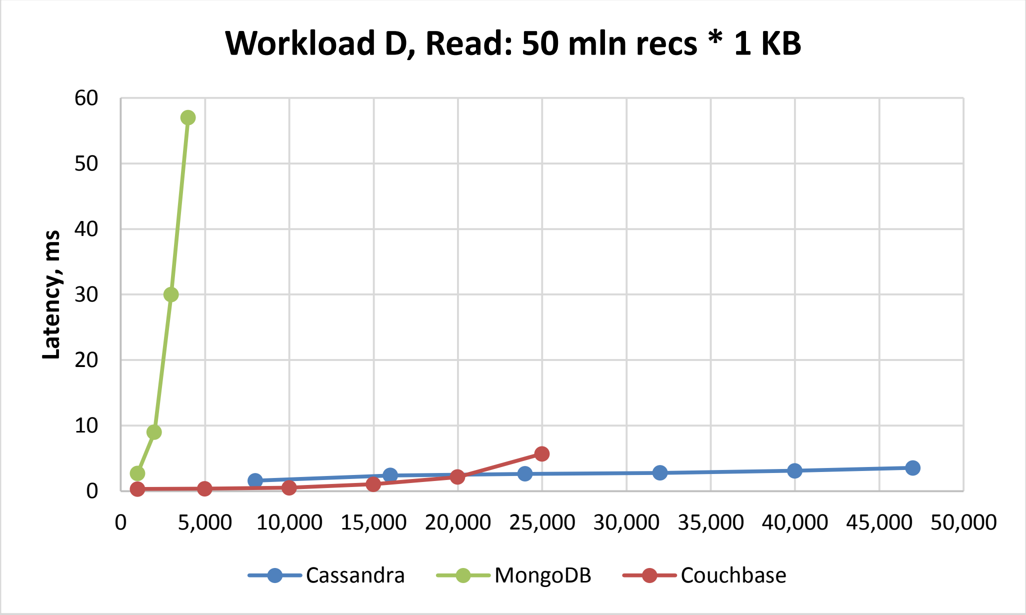
a) Workload D, 50 million records

	Cassandra (DataStax)					MongoDB					Couchbase				
50 mln recs, 1 KB each	29 K, 1.7 ms					5.4 K, 18 ms					87 K, 1.09 ms				
	Op/s	Upd	Del	Ins	Read	Op/s	Upd	Del	Ins	Read	Op/s	Upd	Del	Ins	Read
Workload D	1K					1K	7.5	6.3	5.7	2.7	1K	0.3	0.18	0.23	0.32
	2K					2K	16	13	15	9	2K				
	3K					3K	25	20	30	30	3K				
	4K					4K	37	29	40	57	4K				
	5K					5K					5K	0.35	0.19	0.25	0.36
	8K	0.31	0.29	0.41	1.58	8K					8K				
	10K					10K					10K	0.5	0.23	0.28	0.51
	15K					15K					15K	1.04	0.32	0.38	1.04
	16K	0.36	0.35	0.46	2.38	16K					16K				
	20K										20K	2.1	0.33	0.39	2.15
	24K	0.39	0.37	0.51	2.63	24K					24K				
	25K										25K	5.65	0.45	0.5	5.7
	32K	0.48	0.46	0.6	2.78	32K					32K				
	40K	0.67	0.65	0.79	3.1	40K					40K				
Workload Dmax	47K	0.7	0.68	0.82	3.54										



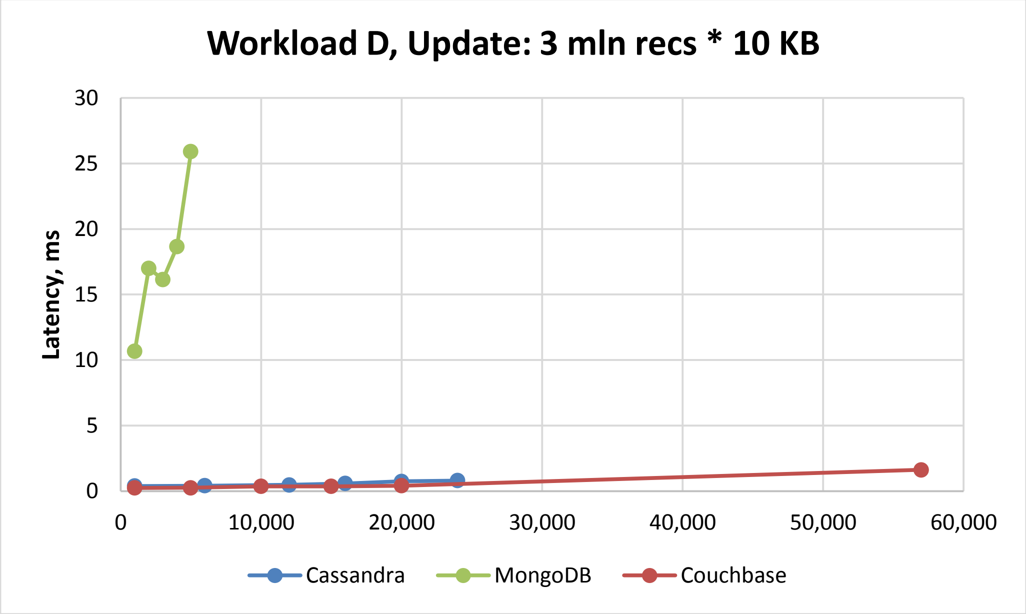

Workload D, Delete: 50 mln recs * 1 KB
Latency, ms
35
30
25
20
15
10
5
0
0
5,000
10,000
15,000
20,000
25,000
30,000
35,000
40,000
45,000
50,000
Cassandra
MongoDB
Couchbase

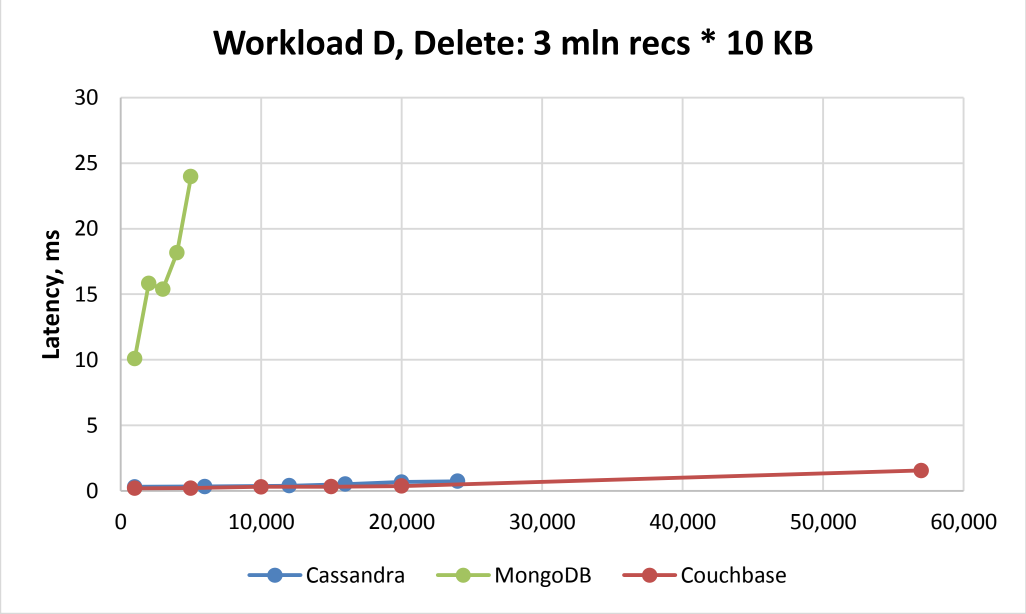

Workload D, Insert: 50 mln recs * 1 KB
Latency, ms
45
40
35
30
25
20
15
10
5
0
0
5,000
10,000
15,000
20,000
25,000
30,000
35,000
40,000
45,000
50,000
Cassandra
MongoDB
Couchbase

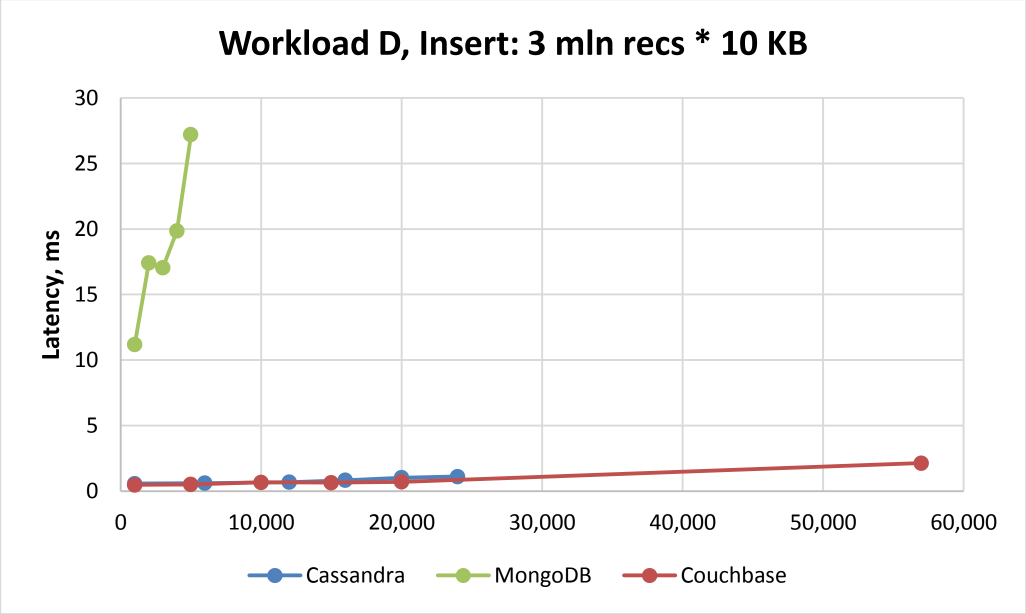


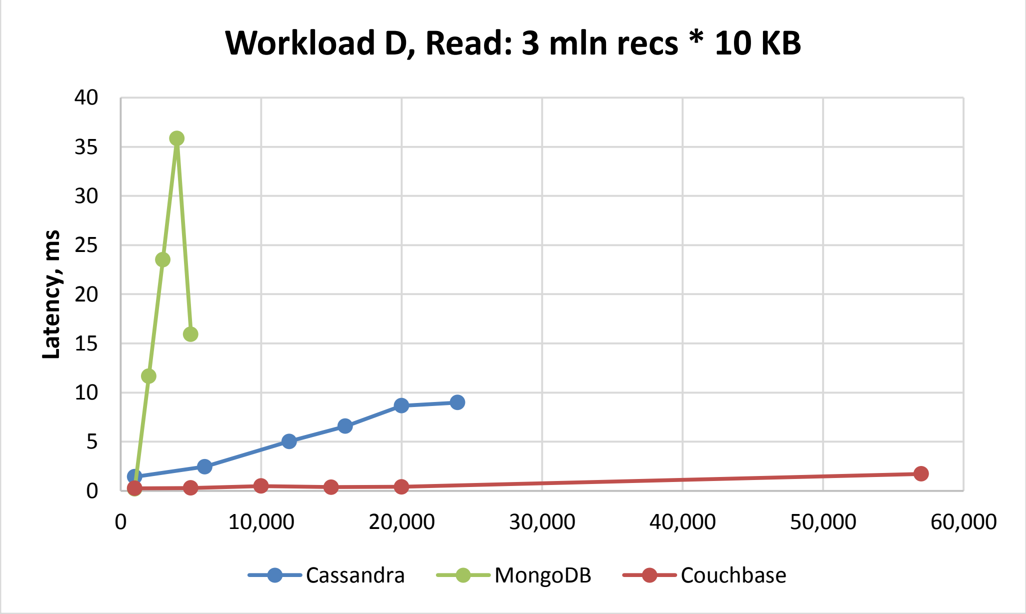
b) Workload D, 3 million records

	Cassandra (DataStax)					MongoDB					Couchbase				
3 mln recs, 10 KB each	**7.5 K, 5.8 ms**					**2 K, 5.09 ms**					**10 K, 9.4 ms**				
	Op/s	**Upd**	**Del**	**Ins**	**Read**	**Op/s**	**Upd**	**Del**	**Ins**	**Read**	**Op/s**	**Upd**	**Del**	**Ins**	**Read**
Workload D	**1K**	0.39	0.31	0.57	1.43	**1K**	10.67	10.1	11.2	0.21	**1K**	0.24	0.19	0.48	0.26
	2K					**2K**	17	15.84	17.41	11.66	**2K**				
	3K					**3K**	16.14	15.39	17.05	23.52	**3K**				
	4K					**4K**	18.65	18.17	19.86	35.85	**4K**				
	5K					**5K**	25.92	23.99	27.21	15.93	**5K**	0.26	0.21	0.51	0.3
	6K	0.42	0.33	0.61	2.46	**6K**					**6K**				
	10K					**10K**					**10K**	0.36	0.3	0.67	0.48
	12K	0.47	0.39	0.68	5.03	**12K**					**12K**				
	15K					**15K**					**15K**	0.37	0.32	0.65	0.38
	16K	0.58	0.51	0.83	6.58	**16K**					**16K**				
	20K	0.75	0.67	1.03	8.66	**20K**					**20K**	0.42	0.37	0.71	0.4
	24K	0.81	0.73	1.11	8.98	**24K**					**24K**				
Workload Dmax											**57K**	**1.62**	**1.55**	**2.13**	**1.72**

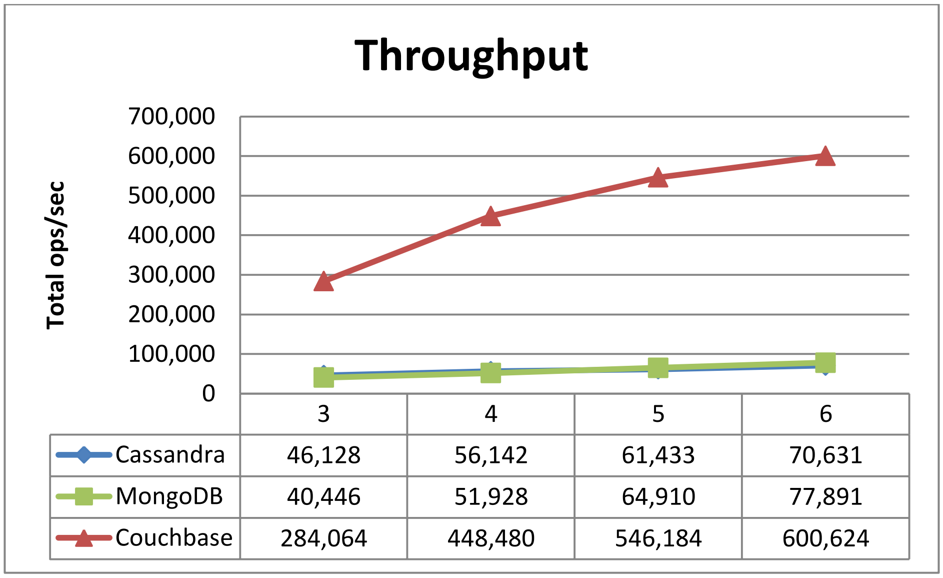

Workload D, Update: 3 mln recs * 10 KB
Latency, ms
30
25
20
15
10
5
0
0
10,000
20,000
30,000
40,000
50,000
60,000
Cassandra
MongoDB
Couchbase


Workload D, Delete: 3 mln recs * 10 KB
Latency, ms
30
25
20
15
10
5
0
0
10,000
20,000
30,000
40,000
50,000
60,000
Cassandra
MongoDB
Couchbase


Workload D, Insert: 3 mln recs * 10 KB
Latency, ms
30
25
20
15
10
5
0
0
10,000
20,000
30,000
40,000
50,000
60,000
Cassandra
MongoDB
Couchbase

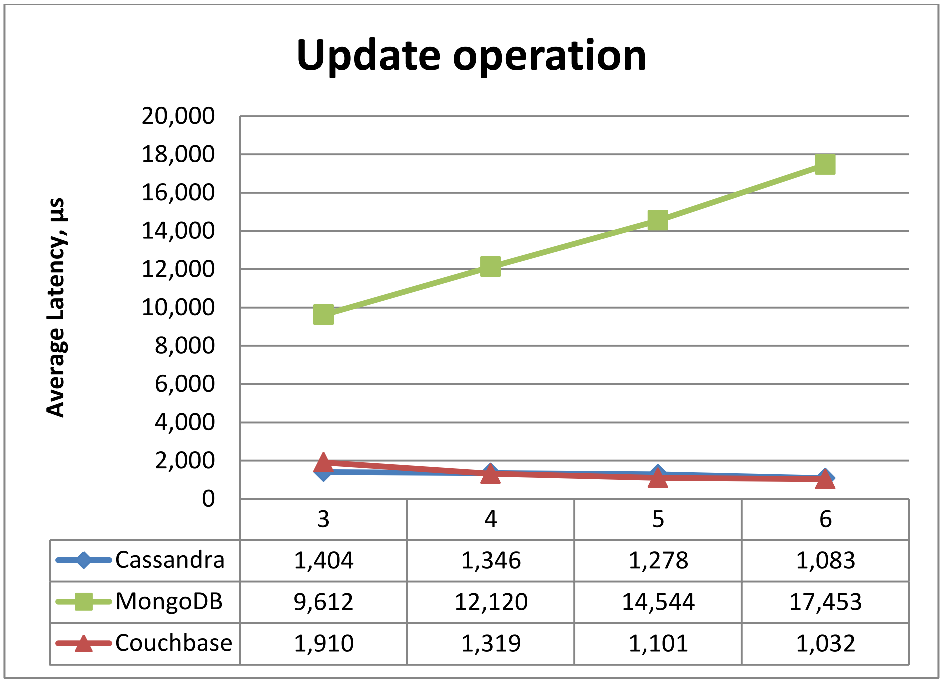

Workload D, Read: 3 mln recs * 10 KB
Latency, ms
40
35
30
25
20
15
10
5
0
0
10,000
20,000
30,000
40,000
50,000
60,000
Cassandra
MongoDB
Couchbase

7. Appendix C: Scalability results

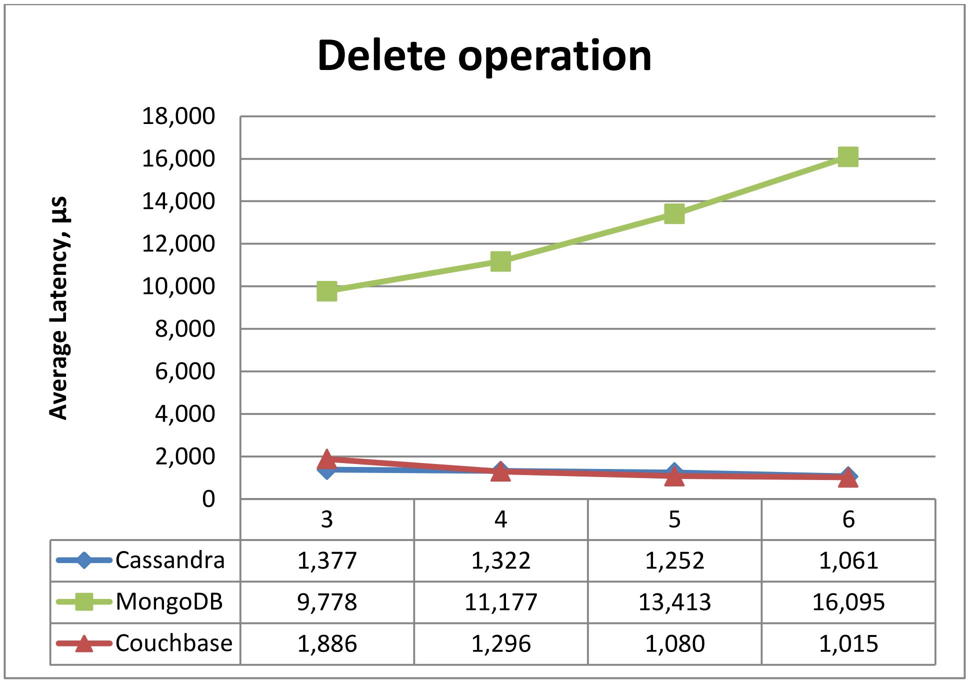


	3	4	5	6
Cassandra	46,128	56,142	61,433	70,631
MongoDB	40,446	51,928	64,910	77,891
Couchbase	284,064	448,480	546,184	600,624

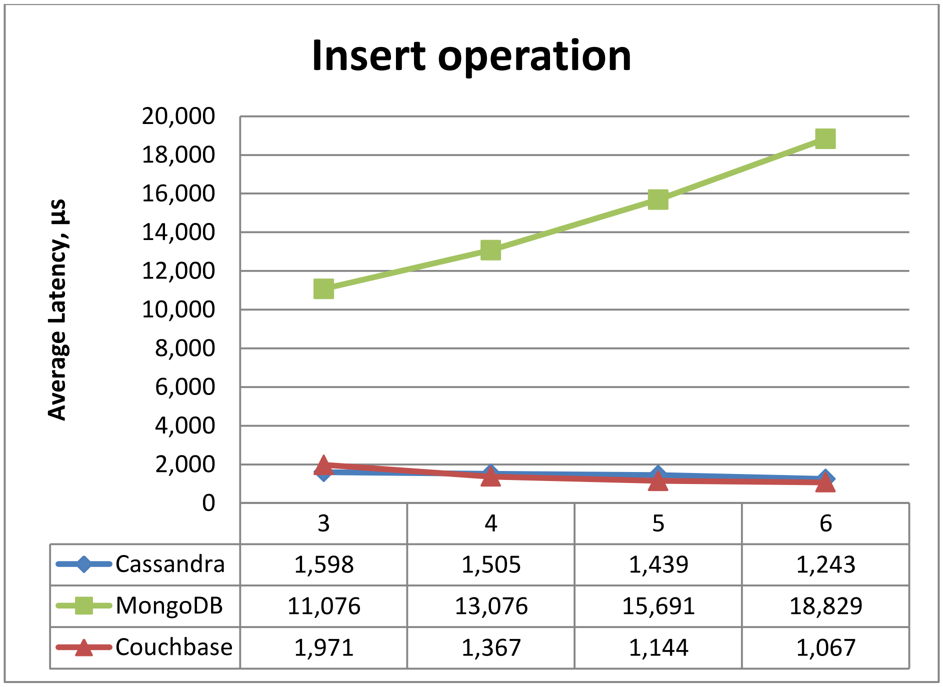
Chart 1, Throughput (ops/sec) vs. number of nodes



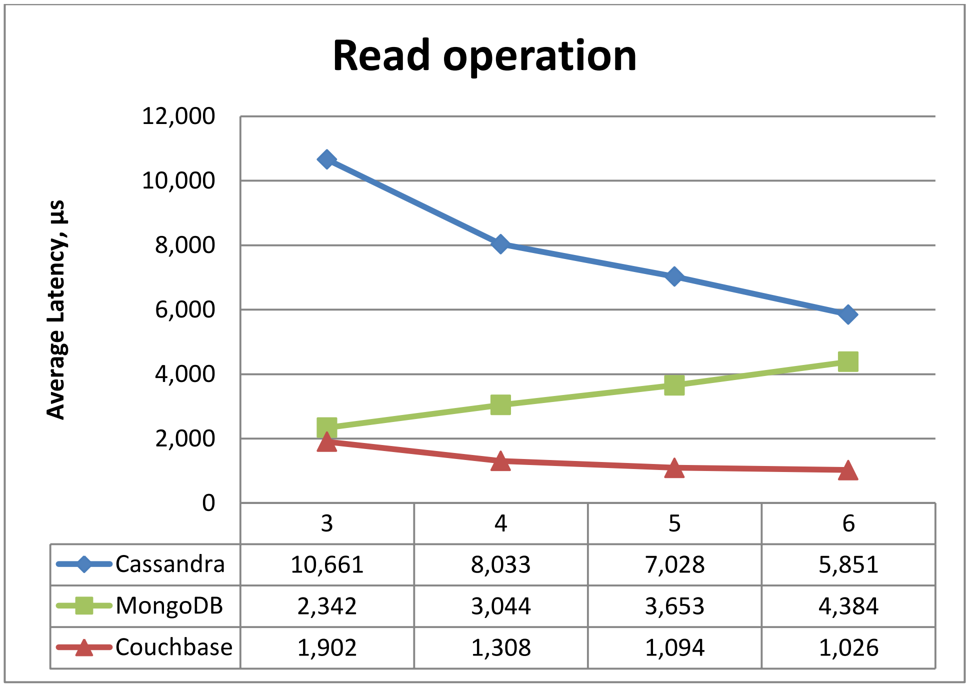
	3	4	5	6
Cassandra	1,404	1,346	1,278	1,083
MongoDB	9,612	12,120	14,544	17,453
Couchbase	1,910	1,319	1,101	1,032



	3	4	5	6
Cassandra	1,377	1,322	1,252	1,061
MongoDB	9,778	11,177	13,413	16,095
Couchbase	1,886	1,296	1,080	1,015



	3	4	5	6
Cassandra	1,598	1,505	1,439	1,243
MongoDB	11,076	13,076	15,691	18,829
Couchbase	1,971	1,367	1,144	1,067



	3	4	5	6
◆ Cassandra	10,661	8,033	7,028	5,851
■ MongoDB	2,342	3,044	3,653	4,384
▲ Couchbase	1,902	1,308	1,094	1,026

8. About the author

Altoros brings Cloud Foundry-based “software factories” and NoSQL-driven “data lakes” into organizations through training, deployment, and integration. With 250+ employees across 8 countries, Altoros is the company behind some of the world’s largest Cloud Foundry and NoSQL deployments. For more, please visit www.altoros.com or follow @altoros.