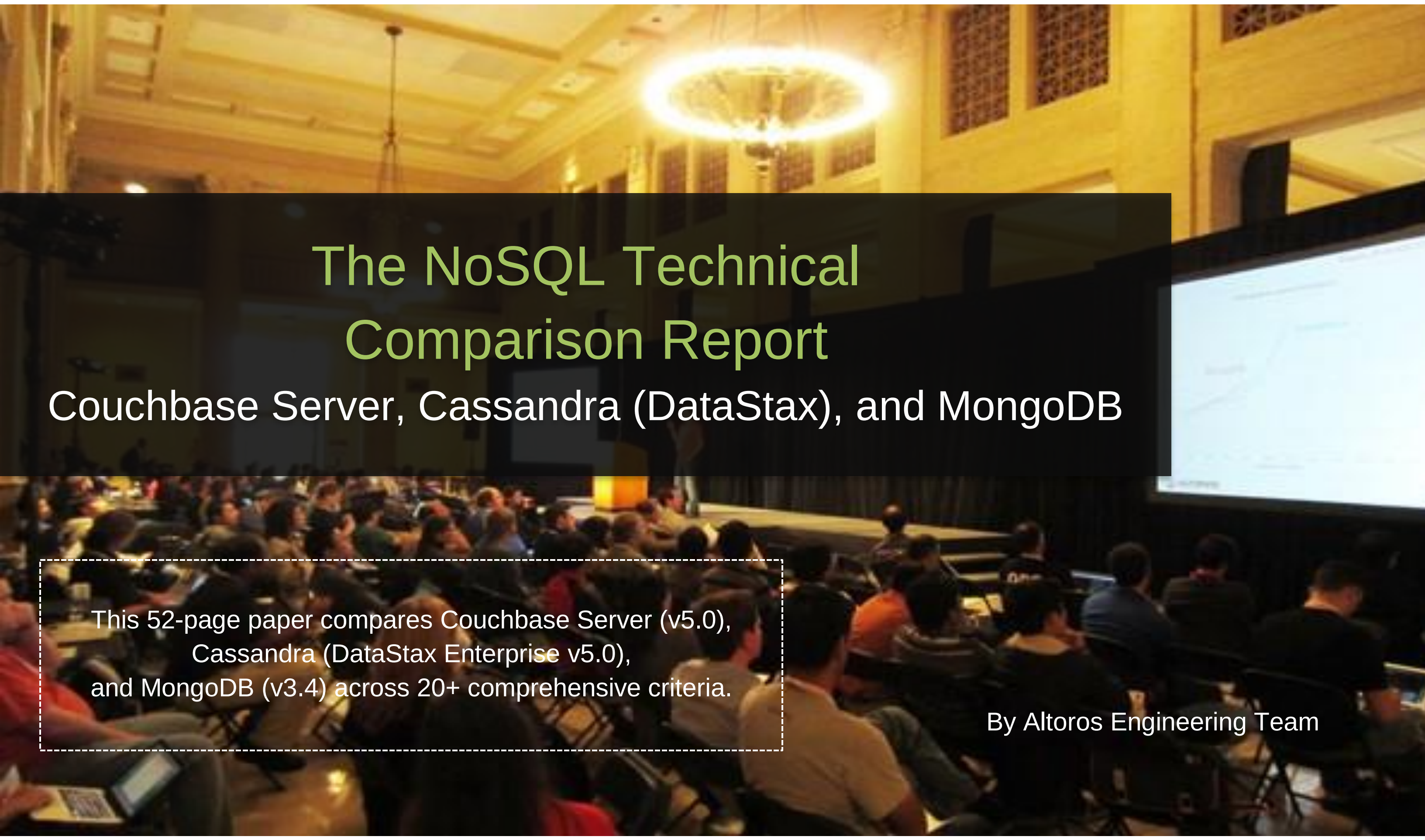
ALTOROS™



The NoSQL Technical Comparison Report

Couchbase Server, Cassandra (DataStax), and MongoDB

This 52-page paper compares Couchbase Server (v5.0), Cassandra (DataStax Enterprise v5.0), and MongoDB (v3.4) across 20+ comprehensive criteria.

By Altoros Engineering Team

Couchbase mongoDB Cassandra

Q3 2017

Table of Contents

1. Introduction

Dozens of NoSQL databases have been developed over the last decades with a goal to deliver faster performance than traditional relational database management systems (RDBMS) in various use cases—most notably those involving big data. However, one should understand that the majority of NoSQL databases are optimized, or even built, for a specific workload or a task. Not all the NoSQL products are the same: implementations vary significantly from vendor to vendor, so it is important to be aware of the comparative strengths and weaknesses of the various NoSQL options available. This is why, before deciding which NoSQL database to use in production, system architects and IT managers typically compare NoSQL data stores in their own environments, using data and user interactions that are representative of the expected production workloads.

This report provides an in-depth analysis of the leading NoSQL systems: Cassandra (DataStax Enterprise v5.0), Couchbase Server (v5.0), and MongoDB (v3.4). 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. 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 the last chapter, 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 data, scores were applied based on real-world practical experience in using the products under evaluation and regularly conducted benchmarks. For criteria based on qualitative data—e.g., installation and maintenance procedure—scores were applied based on in-depth review of the documentation, dialog with the solutions vendors and users engineering teams, and our own development and production experience.

Cassandra is a partitioned row store with the rows organized in tables. Initially designed at Facebook by Amazon Dynamo's developers, it aims to provide high availability and linear scalability. It features tunable consistency, hinted handoff, and active anti-entropy. DataStax Enterprise is a commercial third-party software that provides enterprise features on top of open-source Apache Cassandra, such as full-text search, real-time analytics, advanced security, and auditing.

MongoDB is a web-scale document-oriented NoSQL database. It has extensive support for a variety of secondary indices and API-based ad-hoc queries, as well as strong features for manipulating JSON documents. The database puts forward a separate and incremental approach to data replication and partitioning that happen as completely independent processes.

Couchbase Server is both a JSON document and a key-value distributed NoSQL database. It guarantees high performance with a built-in object-level cache, asynchronous replication, and data persistence. The database is designed to scale out or scale up compute-, RAM-, and storage-intensive workloads independently.

2. Installation and Configuration

This section refers to the ease and convenience of installation and configuration

Couchbase Server

Couchbase Server installation consists of the following 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 the command-line interface (CLI), REST API, or a Web UI.
3. Add all the other nodes to the cluster.

Couchbase documentation contains detailed descriptions of the installation process for Linux, Windows, and Mac OS systems. Mac OS is supported only for developer use. Couchbase Server has a preconfigured Amazon Machine Image (AMI) in the Amazon Web Services (AWS) Marketplace that allows you to launch it in AWS. Couchbase Server Enterprise Edition Google Cloud Launcher enables you to quickly deploy a server on the Google Cloud Platform. The Azure Resource Manager (ARM) template allows you to deploy Couchbase Enterprise on Microsoft Azure. Couchbase Server Docker images are available at the Docker Hub.

Cassandra

Cassandra requires pre-installation of the Java Development Kit. There are three ways to install Cassandra:

1. Using RPM or Debian package manager
2. Employing a binary tarball
3. Manually building binary files from the source code

Cassandra should be installed and configured on each node in the cluster. Configuration could be done through property files or via the DataStax OpsCenter Web UI. Preconfigured templates for Cassandra are provided by major cloud platforms (AWS, Microsoft Azure, Google Cloud Platform) and a Docker registry. Mac OS and Windows are supported only for development.

MongoDB

MongoDB Ops Manager allows for installing and configuring a MongoDB cluster automatically. (A cloud-hosted version is available, as well.) To use Ops Manager, first set up a network. Then, install MongoDB Enterprise dependencies, Ops Manager packages, and Automation Agents on each machine in the cluster. After that, create a database for Ops Manager. Once these processes are completed, you can bootstrap MongoDB cluster through the Web UI.

Manual installation and configuration for a MongoDB sharded cluster is a fairly complicated procedure. In short, you need to satisfy installation prerequisites, then separately configure all the data shards, configuration servers, and sharding routers to finally join those components into a cluster.

Before deploying a production cluster, the database engineers must think carefully about the cluster architecture and answer the following questions:

1. How many primaries (data shards) will the cluster have?
2. How many secondaries and arbiters should each shard contain?
3. What storage engines are going to be used?
4. How many mongoses (the MongoDB sharding process) are going to be run?
5. Where are these mongoses to be run?

Though deployment can be complicated, there are preconfigured MongoDB images available for most cloud platforms, as well as a number of competing Database-as-a-Service offerings.

Summary

Thanks to the Cassandra and Couchbase integrated administration consoles, these data stores can be configured in a single place, making it easier to install them. The MongoDB cluster topology is hierarchical by nature, which makes deployment a bit more complicated than with peer-to-peer data stores. Specifically, you would need to spend more time planning the deployment architecture.

Table 2.1. Ease and convenience of installation on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
8	10	10

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 and different levels of system scalability and availability. This section also contains a brief description of multi-datacenter and multi-cluster deployment topologies.

Couchbase Server

The Couchbase Server topology is flexible. It allows you to configure either a pure homogenous cluster, with all the nodes being equal in responsibilities, or a heterogenous cluster, where different nodes are assigned different responsibilities. Couchbase Server provides several node or service component types that support corresponding workloads within the cluster, including the Cluster Manager, Data, Index, Query, and Search services. Each node runs a mandatory Cluster Manager service. Other services are enabled by default or can be enabled and disabled during a server setup to implement a unique user-defined topology.

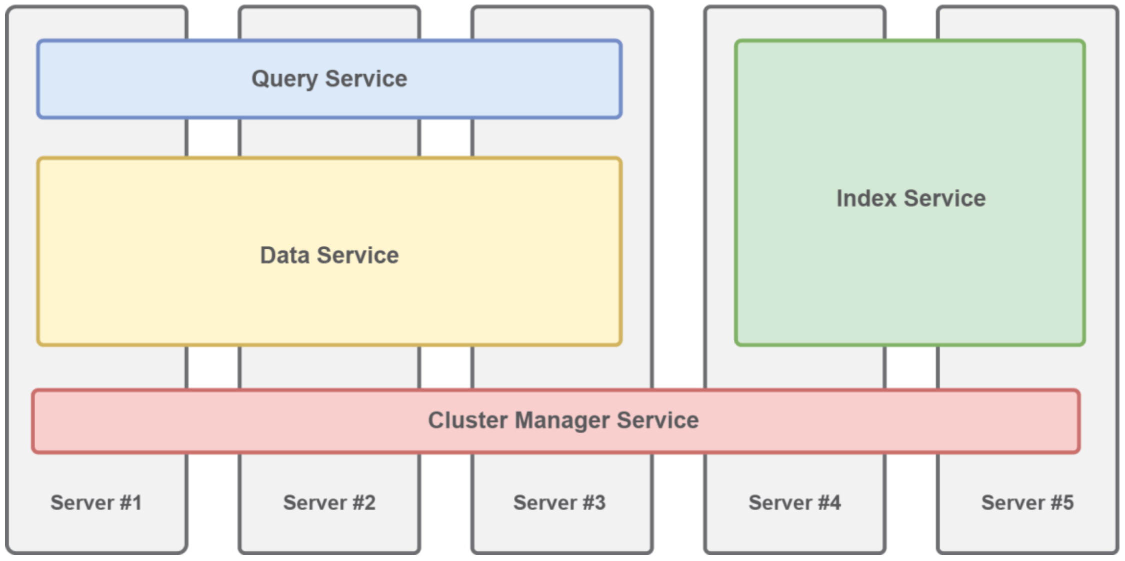


Image 3.1. The Couchbase Server cluster topology

Cluster Manager is responsible for all cluster-wide operations, such as monitoring the cluster state and generating aggregate statistics, performing rebalancing operations to evenly distribute data over the nodes, keeping smart clients synced with the current cluster topology, handling authentication and logging, etc. The Data Service is a core system component that handles key-value data store and incremental MapReduce views. These key-value pairs (documents) are hash- and rack-aware and distributed across all the Data Service nodes. The Query Service is responsible for handling SQL-like queries (N1QL). The Index Service indexes data through the Global Secondary Indexes (GSI) to support the query service. The Search Service provides full-text search support. All the cluster components communicate through the Database Change Protocol (DCP), which was designed to quickly and efficiently move large amounts of data.

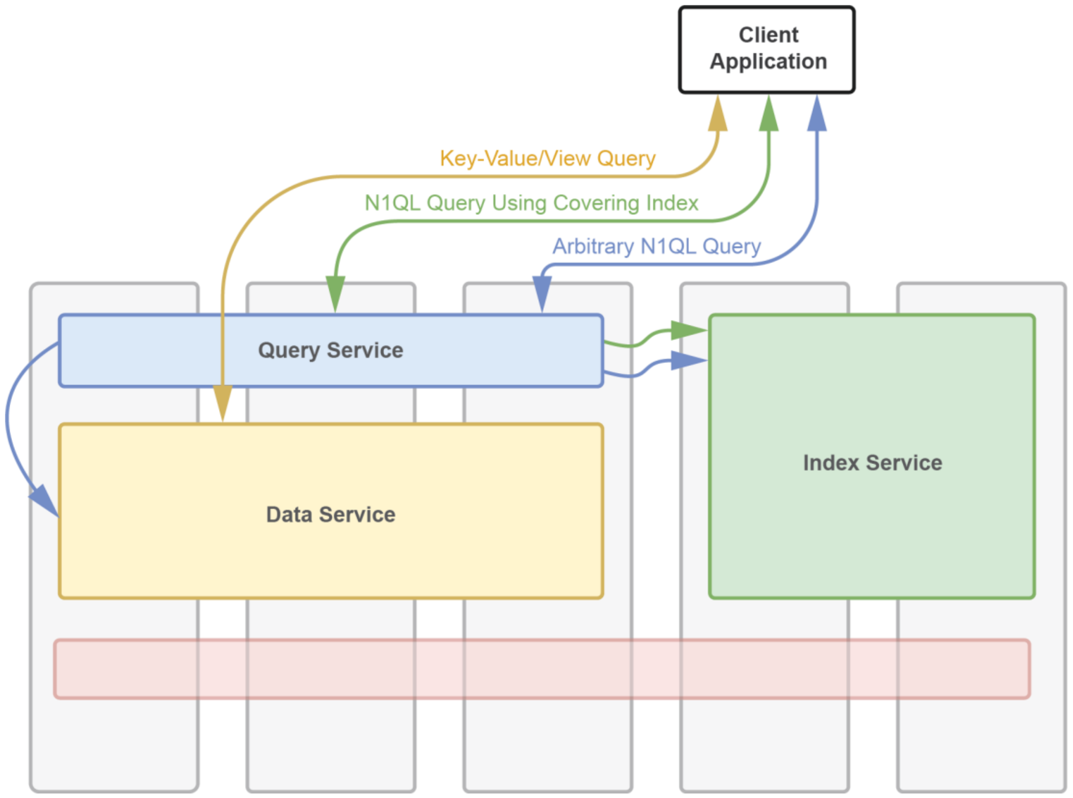


Image 3.2. The Couchbase Server query path

The cross datacenter replication (XDCR) allows for setting up data replication between two specified buckets—source and destination—located in remote clusters. The feature enables both unidirectional (active-standby) and bidirectional (active-active) replication for centralized, hierarchical, ring, or any user-defined multi-cluster topologies.

Cassandra

Cassandra is a distributed cluster system consisting of homogeneous nodes that form a peer-to-peer topology. Data is evenly distributed across all the nodes. Cluster nodes communicate with each other using the *gossip* protocol. Gossip is used to exchange nodes' state information about themselves and about other nodes they know. Some nodes must be communicated with. These “seed” nodes prevent distributed system logical partitioning by eventually reconciling membership of other nodes. Data center and rack awareness are provided by the *snitch* process, which enables efficient dynamic request routing and smart replica distribution.

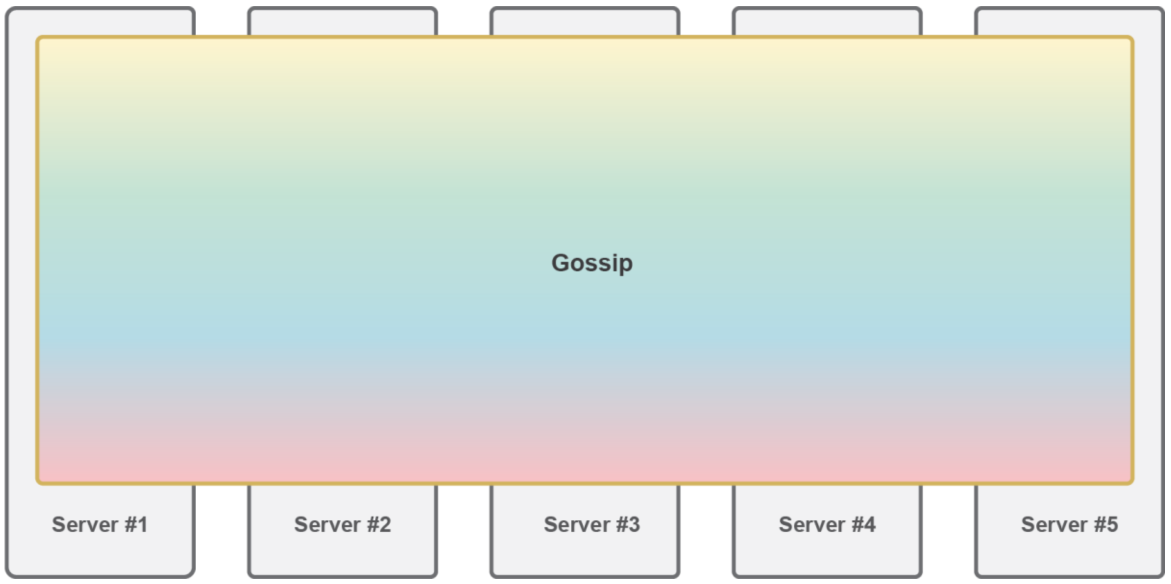


Image 3.3. The Cassandra topology

DataStax Enterprise integrates various external open-source components to support full-text search and analytics on top of a single persistence layer. Those components are deployed on separate nodes.

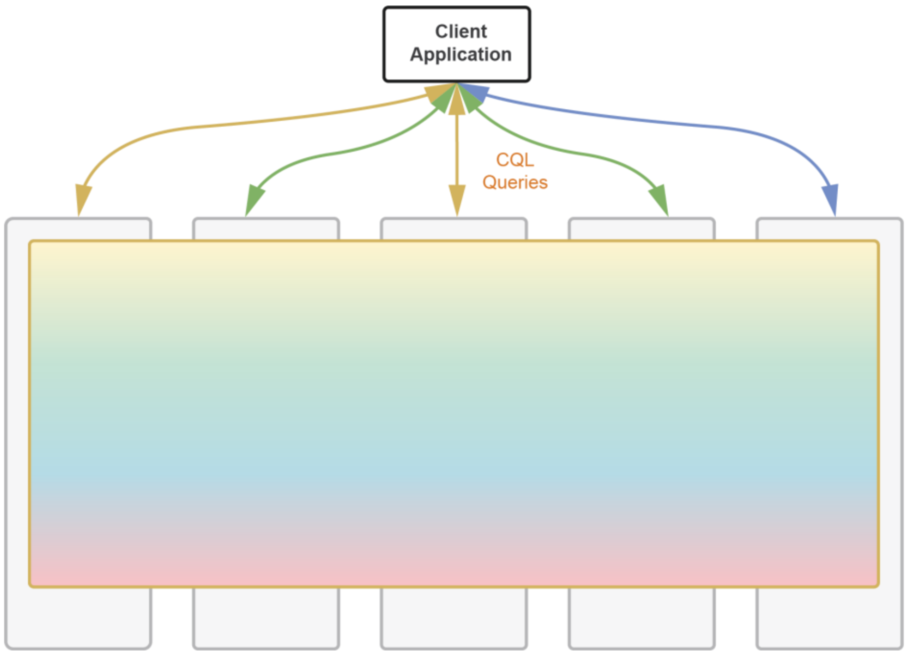


Image 3.4. The Cassandra query path

Active-active and active-standby topologies can be configured using flexible Cassandra features dedicated to *multi-datacenter* cluster deployments: a snitch, tunable consistency, a replication factor, and a strategy. Advanced replication by DataStax is used for *multi-cluster* deployments. It leverages source and destination cluster replication channels to build both common (active-standby or active-active) and more sophisticated topologies (hub-and-spoke or mesh networks).

MongoDB

MongoDB employs a hierarchical cluster topology that combines router processes, configuration servers, and data shards. Clients do not directly communicate with data shards, as all the requests must be proxied by router nodes (the “mongos” processes). The router nodes retrieve, cache, and use the sharding (data distribution) information from the configuration nodes. These configuration nodes, called “config servers,” store metadata that reflects the state and organization for all the data and components within the cluster. A number of data shards together with config servers and mongoses are combined in a sharded cluster. Both config servers and data shards are typically deployed as replica sets for redundancy.

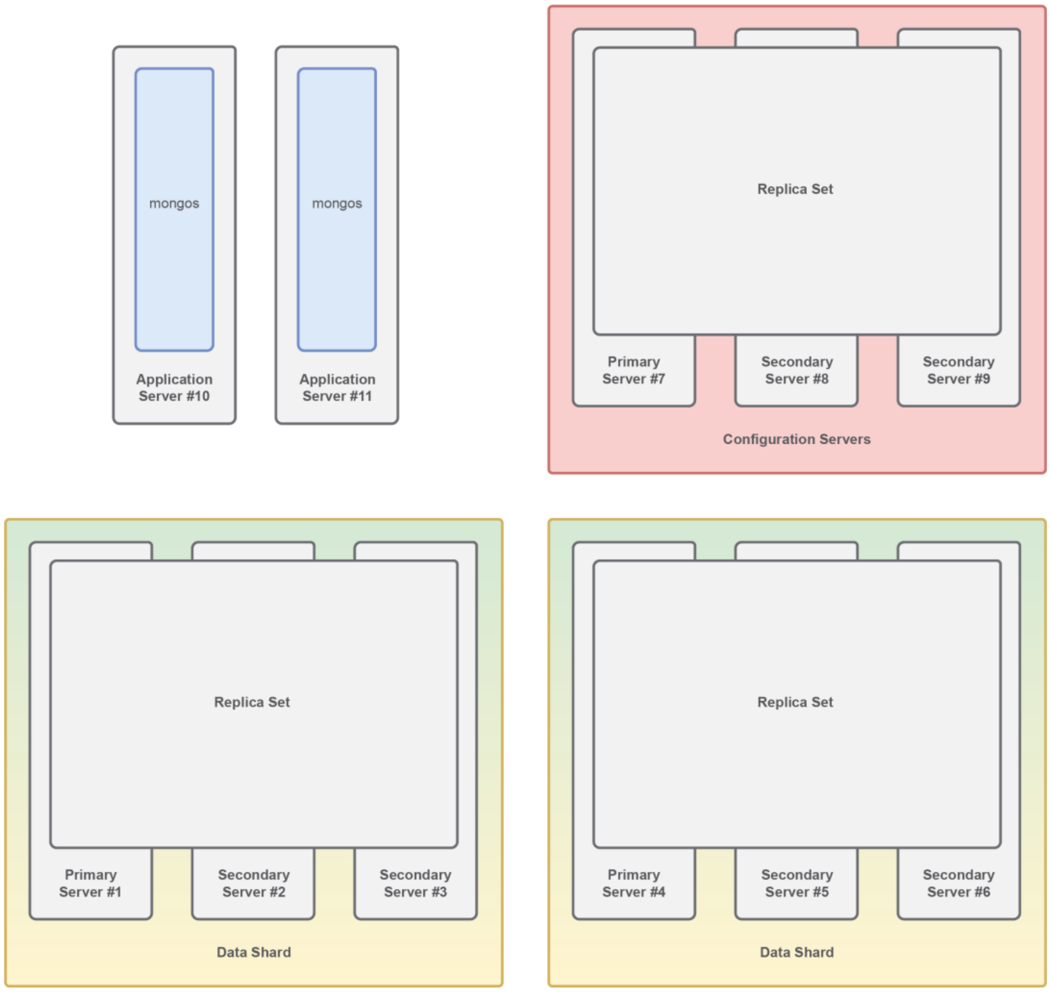


Image 3.5. The MongoDB cluster topology

A replica set usually includes a single primary node, multiple secondary nodes, and an optional arbiter node. The relationship between the primary and secondary nodes may be described as a specific case of the master-slave replication. All the data mutation operations are served exclusively by a single master—the replica set primary node. If the current primary goes down, elections take place. One of the secondaries becomes the new primary and continues to accept write operations. The arbiter node participates in the elections to break possible election ties. So, if a replica set has an even number of nodes, one must add an arbiter to a replica set for the purpose.

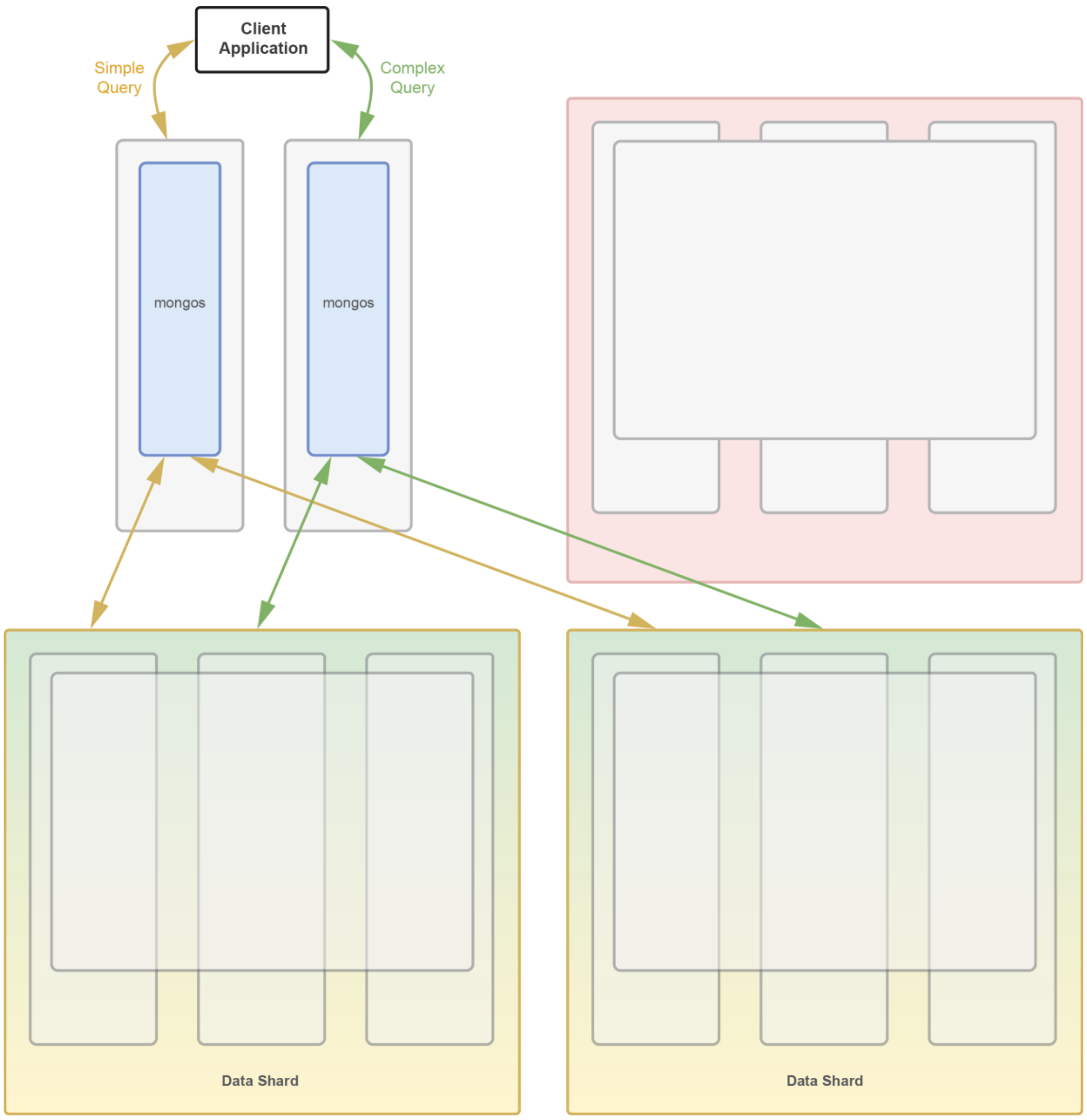


Image 3.6. The MongoDB query path

MongoDB cluster replica set members can be distributed across multiple data centers in order to support multi-datacenter deployments. The “nearest” read preference option allows data to be served from the closest replica set member based on a ping distance to a user. Such reads are eventually consistent as the replica set secondaries stay behind the primary due to both usual replication latency and additional cross-datacenter network latency. Data mutations are issued to the primaries that can be either located in the main data center to support active-standby pattern or distributed evenly across multiple data centers for the write workload equalization.

Tag aware sharding allows users to control data distribution in a multi-datacenter MongoDB cluster by tagging ranges of a shard key to one or more data shards. The resulting specific deployment architecture together with the reinforced application-side logic allows to implement distributed local writes.

Summary

Cassandra nodes form a pure peer-to-peer homogeneous cluster. The resulting cluster does not have a single point of failure; most of the cluster-wide services are hidden from its clients and administrators; it is horizontally scalable and simple. The Couchbase Server cluster structure has similar design origins—all the nodes are equally important. At the same time, its multidimensional scaling feature enables granular control over workload segregation as an alternative to the pure equal workload distribution. MongoDB advocates for multi-datacenter clusters over multi-cluster installations. In contrast, Couchbase Server leverages its XDCR to empower multi-cluster deployments. Cassandra has a strong support for both multi-datacenter and multi-cluster topologies. In comparison to Couchbase Server and Cassandra, both single and multi-datacenter MongoDB deployment architectures are more complex and less encapsulated, providing no significant benefits.

Table 3.1. Topology of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
7	9	9

3.1.2 Scalability

Scalability is the ability of a distributed database system to handle a growing amount of data it stores and processes. The ways the system may be enlarged to accommodate that growth include horizontal and vertical scaling, or the combination of both. Horizontal scaling of a distributed database, or scaling out, refers to a strategy of increasing the overall system capacity by adding more nodes to the cluster. Vertical scaling, or scaling up, means adding more resources to the nodes in a cluster. In homogeneous topologies, you typically have to add resources to each node in the cluster, whereas in heterogeneous topologies, you add resources only to the nodes that support the specific services you want to increase.

Couchbase Server

Couchbase Server comprises a number of services—Data, Index, Query, and Search—handling different workloads. Each service has its own system resource requirements. Multidimensional scaling (MDS) enables independent expansion of these services and the corresponding workloads both vertically and horizontally. For example, if a workload requires a big volume of queries or a lot of complex queries, you can add a node for the Query Service without creating side effects for other cluster services. The standard symmetrical scaling is also available. It gives each node an equal share of work for all the services with a potential contention for computational resources between them.

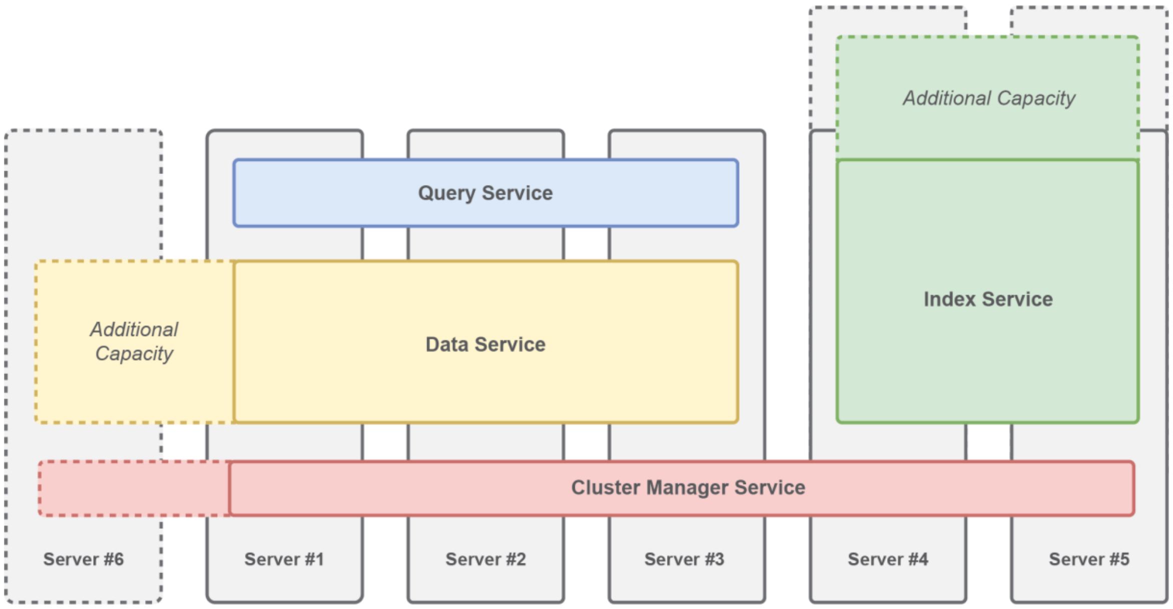


Image 3.7. The Couchbase Server multidimensional scaling

Each service scales both vertically and horizontally but there are some differences between them from the optimal resource perspective:

- The Data Service is optimized for RAM and disk access with low CPU requirements. It performs best with numerous small nodes rather than a few large ones.
- The Query Service is CPU-intensive and has very low RAM and disk requirements. There is not much difference between many small nodes or fewer large nodes from the performance perspective. This service is stateless and can be scaled in and out very quickly because no data is moved when adding or removing query capacity.
- The Index and Search services are optimized for RAM and disk access, with CPU requirements driven by the incoming write load. These services perform best with fewer, larger nodes.

Cassandra

Cassandra features almost linear horizontal scalability. Capacity can be increased by adding more nodes to a cluster. When a node joins a cluster, it acquires responsibility for an even portion of data from other nodes. On the contrary, if a node fails, the load is spread evenly across other nodes in a cluster. The Full Text Search and Data Analytics services provided by DataStax are scaled separately from the original Cassandra nodes.

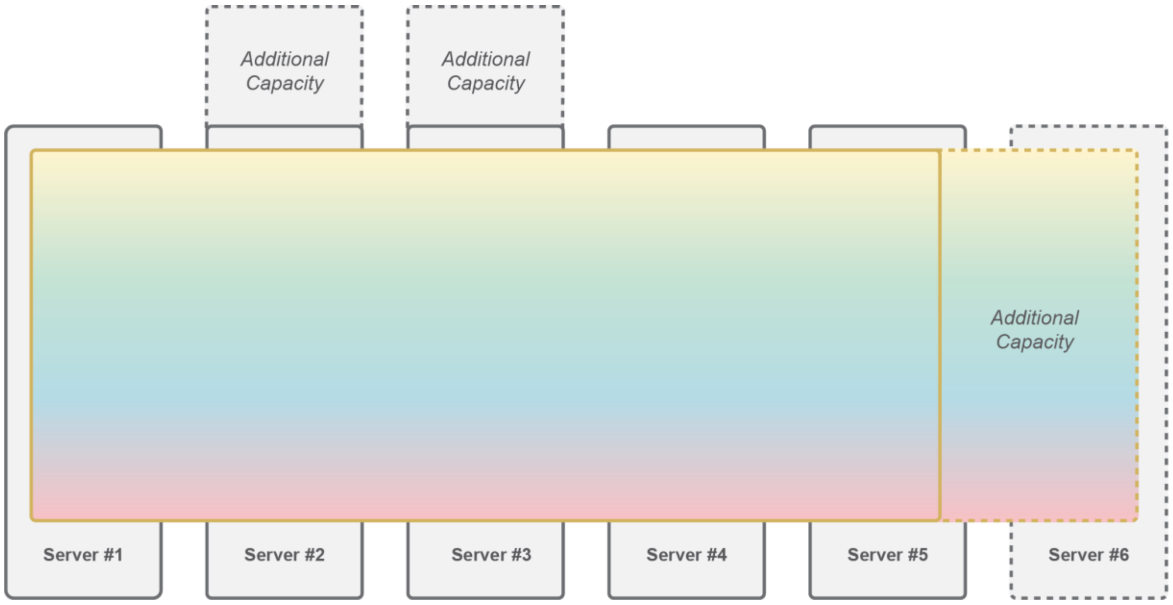


Image 3.8. Scaling in Cassandra

It is recommended to allocate less than 32 GB of the main memory for a JVM heap to employ compressed references optimization for 64-bit architectures. Other allocated memory is used for OS file-system and off-heap caches. Cassandra performs lots of intensive parallel computing, so multi-core CPUs are also recommended. Local hard drives are preferable over network attached storages.

MongoDB

MongoDB has two complementing strategies for scalability:

1. Scaling the read operations by increasing the number of secondaries in each replica set (a data shard)
2. Scaling both reads and writes by increasing the total number of data shards

To implement the first strategy, a group of mongod processes maintains the same data set and supports master-slave replication. Write operations are always performed on a replica set primary, but the reads are allowed to be performed against the replica set secondaries. By adding more nodes to a replica set, you can achieve almost linear scalability for read operations.

The second strategy is the direct use of sharding—a method for horizontal data partitioning in MongoDB. Data collections in a cluster are sharded explicitly on demand and, potentially, this enables additional flexibility. However, in practice, this type of cluster configuration increases system complexity. Therefore, typically, data shards are kept equal in size.

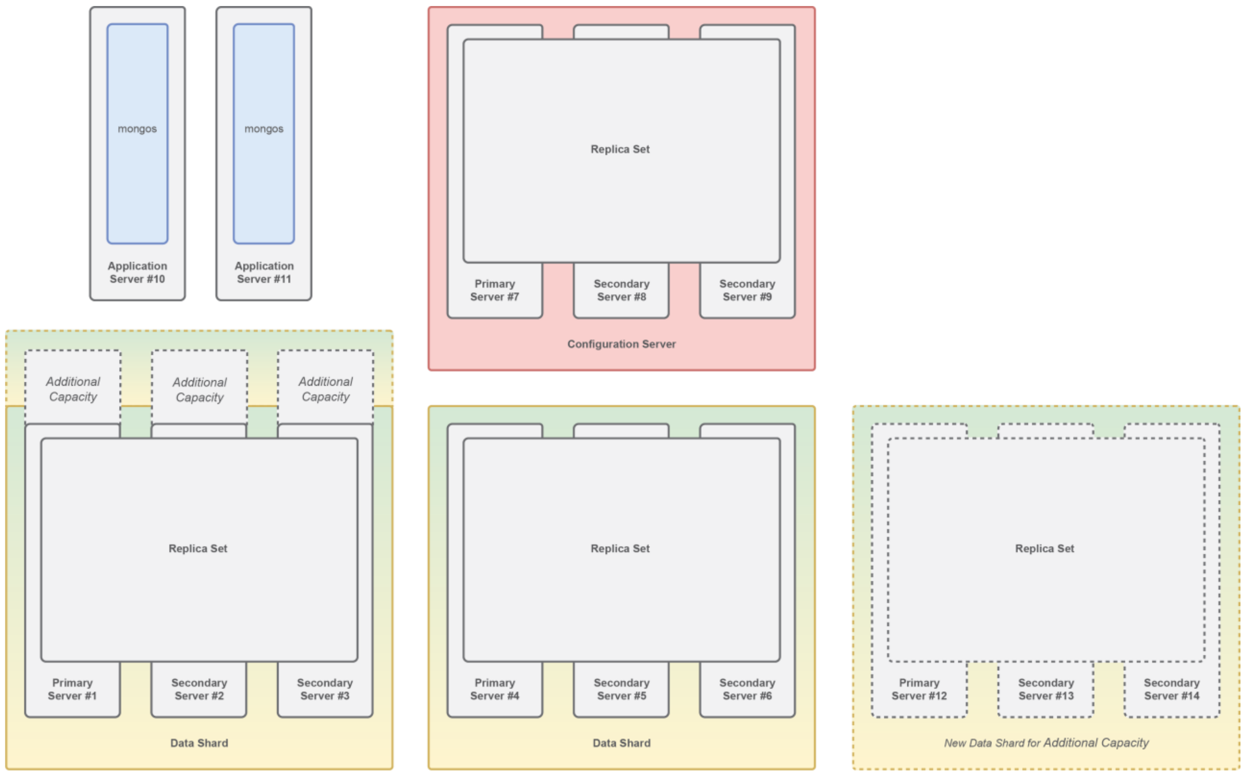


Image 3.9. Scaling in MongoDB

As MongoDB does not imply any restrictions to each mongod process configuration within a replica set, you can combine different storage engines across a single data shard. For example, you might keep one or more replica set members with the in-memory storage engine for high performance and one or more replica set members with WiredTiger for persistence.

Summary

Couchbase Server and Cassandra are easy to scale and do not require any additional actions over a cluster. In case of MongoDB, you have to scrupulously elaborate on the initial cluster design to achieve the same ease of scaling offered by Couchbase Server and Cassandra.

Table 3.2. Scalability of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
8	10	10

3.1.3 Consistency

The data consistency model specifies a contract between the programmer and the system by which the result of the concurrent database operations will be predictable. Database consistency can also refer to the case of the server node and network failures.

Couchbase Server

When data is accessed by a document key, Couchbase Server provides strong consistency. It is achieved by design: each active vBucket exclusively serves all the requests to the subset of keys it maintains. Single or multiple node failures do not affect strong data consistency because a replica vBucket is not responsible for serving writes. It is possible to read data from a replica vBucket but with no strong consistency guarantees. Couchbase XDCR is eventually consistent because of the asynchronous nature of replication between Couchbase Server clusters in remote data centers. The system provides two conflict resolution strategies:

- The timestamp-based resolution—"the most recent update wins"
- The revision-based strategy—"the most updates wins"

GSI, view indexes, and full-text search indexes are eventually consistent as mutations are streamed and processed by the indexers asynchronously. Although the indexes are updated asynchronously, strong query consistency is achieved by applying a parameter on a per-query basis (*stale* on view queries and *scan_consistency* on the N1QL and FTS queries). Done on a per-query and per-index basis, the average performance of the whole system does not need to suffer when enforcing strong consistency for the individual queries.

Cassandra

According to the Brewer's CAP theorem (which highlights the trade-offs in a distributed data store among consistency, availability, and partition tolerance), Cassandra is typically considered to be an AP system, providing high availability (A) and partition tolerance (P). At the same time, it could be configured as a CP system using a tunable consistency (C) level available for both read and write operations. For write operations, the tunable consistency level specifies how many replicas should store data before the request is considered to be successful. For read operations, the read consistency level specifies how many replicas must respond to a read request before returning the result. Thus, to achieve strong consistency in a Cassandra cluster, the client application must use read and write consistency levels, so that *(nodes_written* + *nodes_read)* > *number_of_replicas*. This formula means that strong consistency can be achieved only if the number of the failed replicas is less than half of all the replicas. Otherwise, only eventual consistency can be guaranteed.

Secondary indexes are stored on the same node as source data, so they are fully consistent with the original data. *Materialized views* are globally distributed and updated asynchronously, so they are eventually consistent with the original data.

Inconsistency between replicas occurs in case of *(nodes_written* + *nodes_read)* <= *number_of_replicas*. To resolve it, Cassandra uses the manual and automatic read repair process. During this process, replicas exchange data with each other, and the version of data with the most recent timestamp is stored on all the replicas.

For reconciling concurrent updates of the same data, the "last write wins" approach is used. This approach means that Cassandra provides only read uncommitted isolation by default. *Lightweight* *transactions* can be used to provide serializable isolation and linearizable consistency at a cost of performance.

MongoDB

By default and during regular functioning, the primary MongoDB node is targeted with all the reads and writes designated to its replica set. It means that data is fully consistent. Read operations might also be specifically or optionally targeted to the replica set secondaries via configuring the read preference option. The default write concern for this case guarantees only eventual consistency. (The write concern option specifies how many nodes the primary will wait for before acknowledging a write operation.)

The user might want to balance write concern options, read preference options, and the resulting behavior of the system. However, reasoning about data consistency in case of the replica set primary failure, possible network partition, and other probable malfunctions is fairly tricky and requires outstanding knowledge in the field of distributed systems. Careless experimenting could result in document updates loss, stale and dirty reads, and other problems. Using the non-default *majority* write concern and *linearizable* read concern is generally recommended to prevent possible system inconsistency.

Indexes are synchronously updated within all the data mutation operations. This approach helps to ensure full consistency of the indexes with the original data at a cost of the write performance.

Summary

While Couchbase and MongoDB provide strong consistency by design, Cassandra guarantees eventual consistency and can provide strong consistency only with certain limitations. All the three NoSQL solutions give their users the ability to balance between a consistency level in case of possible cluster temporal malfunctions at a cost of the increased latency.

Table 3.3. Consistency of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	9

3.1.4 Availability

Availability refers to the capability of accessing a database cluster even if a single or multiple nodes goes down. In other words, it is the estimated fraction of attempted operations that succeed during the entire system lifetime or a percentage of uptime in a given year.

Couchbase Server

In terms of the CAP theorem, Couchbase Server is CP, favoring consistency and partition tolerance over availability.

Within the Data Service, documents are hashed across 1,024 subsets/partitions known as vBuckets. While replicated multiple times across the cluster, each vBucket is mastered on a single node at a time to enforce strong consistency. If a node within the Data Service fails, the active vBuckets (and their associated documents) on this node are temporarily unavailable for reads and writes. During this time, individual key-value operations may request replica reads with potentially stale results, but writes will fail until that vBucket is made active again either by a manual/automatic failover or a problematic node recovery.

Currently, the N1QL queries do not have the ability to read from replicas, so they will experience failures if they require reading from the Data Service. Thus, a single Data Service node failure can cause an unavailability of a range of the N1QL queries. It can be mitigated by using covering indexes that can be partitioned and stored separately from data. The Query Service itself is automatically load-balanced via the SDK. So, if a node in this service fails, only currently running queries are affected and can be immediately retried.

The Index and Search services provide high availability via index replicas. In case of a node failure, the replica indexes will be automatically load-balanced even while the failed node is still within the cluster. To remove it, a manual failover is required.

Cassandra

High availability is a core principle of the Cassandra design and implementation. The homogenous peer-to-peer cluster topology ensures the system’s strong partition tolerance with no single point of failure. Replication enables serving writes and reads for the same portion of data on redundant nodes. In case of an arbitrary node failure, other nodes continue to operate properly without any system downtime. It is possible to lose up to a number of replicas minus one and all the cluster data will still be available. Multi-datacenter replication ensures high availability in case of the entire data center failure.

MongoDB

A MongoDB cluster provides high availability for each replica set independently by performing automatic failover through elections for the replica set with a failed node. It allows the replica set secondary member to become a new primary node if the old primary is unavailable. However, the replica set data would not be accessible for mutations during the voting process, which lasts for a few seconds. You can explicitly specify a call for the replica read operation, which does not guarantee data consistency and may lead to a stale read (depending on the write concern used). The failover process itself does not require manual intervention.

Config servers are also deployed as a replica set to tolerate failures of different types of cluster nodes. However, if the config server replica set loses its primary and for any reasons cannot elect a new one, the whole cluster enters the read-only mode. If all the config servers or mongoses are unavailable, the cluster becomes inoperable. Thus, special attention must be paid to the deployment design of config servers and mongoses.

Summary

Cassandra provides high availability out of the box, since it is implied by its architecture and does not require any extra time for recovery. In contrast to MongoDB, Couchbase Server is more robust due to its peer-to-peer nature. However, Couchbase Server allows only a single node to be automatically failed over at once and requires manual intervention in case of a multi-node failure.

Table 3.4. Availability of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
6	8	10

3.1.5 Replication

Replication is the process of copying and maintaining data in multiple physical and/or logical places. It provides high availability and may improve database access performance.

Couchbase Server

Couchbase Server supports data replication both within a single cluster and between multiple ones.

The intra-cluster replication allows for having up to three replicas per bucket. It maintains the specified number of active vBuckets copies intelligently distributed across available cluster nodes. Each data mutation is served in memory by an active vBucket and asynchronously pushed to the replicas via the DCP protocol. The process of implementing data persistence is asynchronous, too. A Couchbase Server client can tune the write durability level with the “persisted to” and “replicated to” options, specifying the number of nodes the data mutation was saved on (written on a disk) and replicated to (residing in RAM) before the write call is acknowledged. By default, replicas are not accessed by the client and strong data consistency is guaranteed. Eventually, consistent replica reads are available in case of a failure, but are not necessary for performance and load balancing reasons. The Full Text Search and Index services have the similar replication mechanisms.

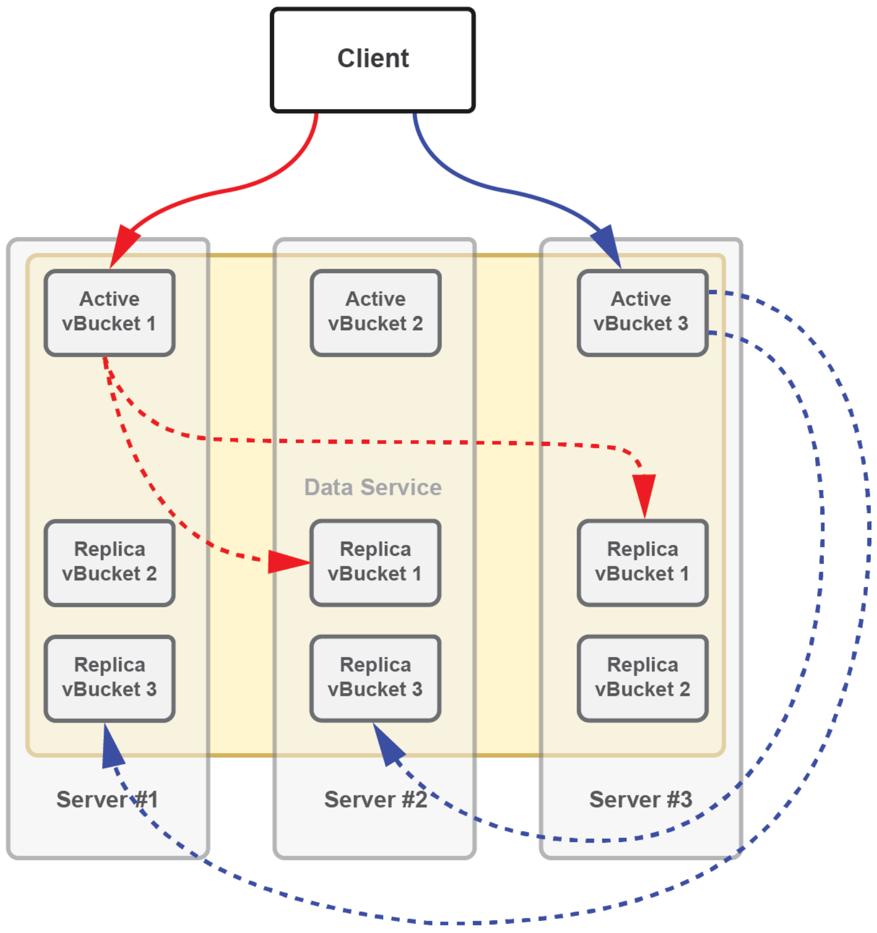


Image 3.10. The intra-cluster replication in Couchbase Server

XDCR allows you to set up data replication between clusters located in different data centers. This type of replication is configured unidirectionally and separately for each bucket. A user specifies the source and destination buckets in the corresponding Couchbase Server clusters. A regular expression for filtering document keys can be specified, and only those keys matching the expression will be replicated. Bi- and multi-directional XDCRs are implemented as a set of unidirectional replications. After the replication is configured, data is asynchronously streamed from the original active vBuckets of one cluster to the target active vBuckets of another. Compared to the intra-cluster replication, mutations are intelligently batched to optimize the network utilization. The XDCR process can be suspended at an arbitrary point in time due to the occurred network failure or configured schedule and resumed from that point later.

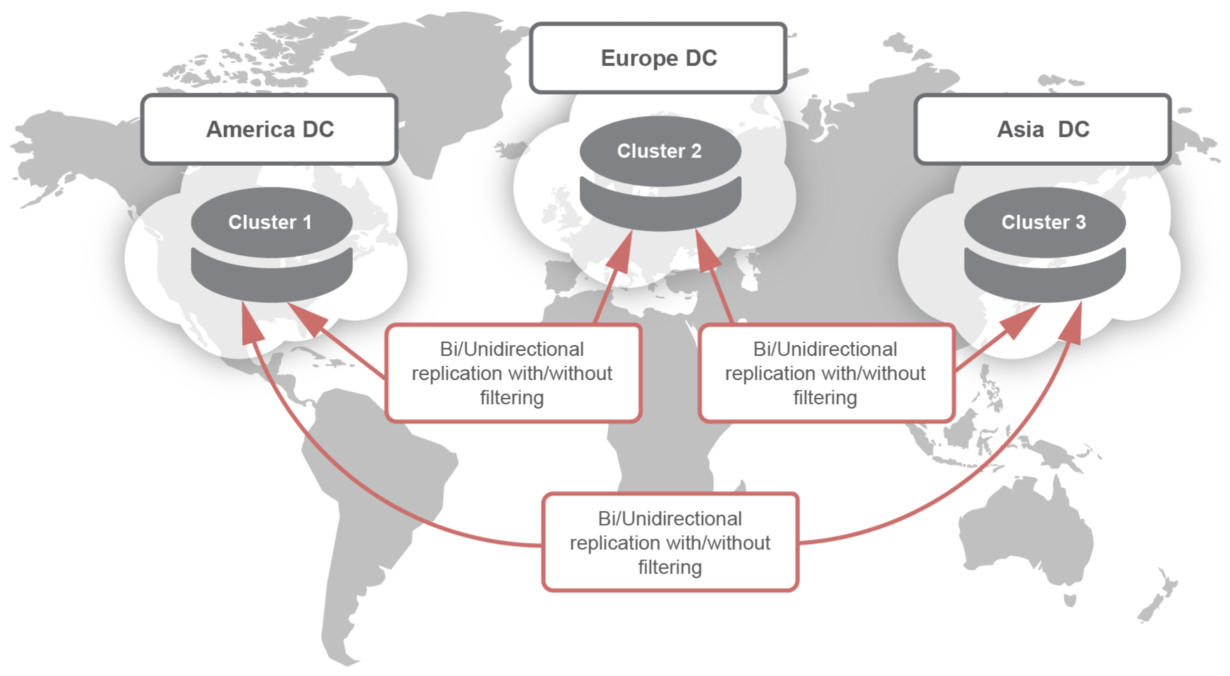

Europe DC
America DC
Asia DC
Cluster 2
Cluster 1
Cluster 3
Bi/Unidirectional replication with/without filtering
Bi/Unidirectional replication with/without filtering
Bi/Unidirectional replication with/without filtering

Image 3.11. The cross datacenter replication in Couchbase Server

The XDCR's configuration can support the mesh, ring, hub-and-spoke, tree, and follow-the-sun topologies, as well as custom topology combinations. This grants users the flexibility to implement a wide variety of topologies and strategies to meet their applications needs.

Cassandra

Cassandra stores replicas on multiple nodes to help ensure reliability, high availability, and partition tolerance. The *replication strategy* specifies how replicas are distributed within a cluster. Two strategies are available: one for single-datacenter deployments, and the other for multi-datacenter deployments. Each data record is distributed across the cluster based on its token. The token is derived from the record's partition key by applying a hash function to it. Multiple ranges of tokens, called *vnodes*, are located on each node.

The replication factor defines the number of nodes responsible for each vnode. Consistent hashing and vnodes are used to distribute records evenly across all the nodes at fine granularity. A tunable consistency level specifies how many replicas should persist the record mutation before the request is considered to be successful. If the tunable consistency level is lower than the replication factor, the record mutation is propagated to other replicas asynchronously.

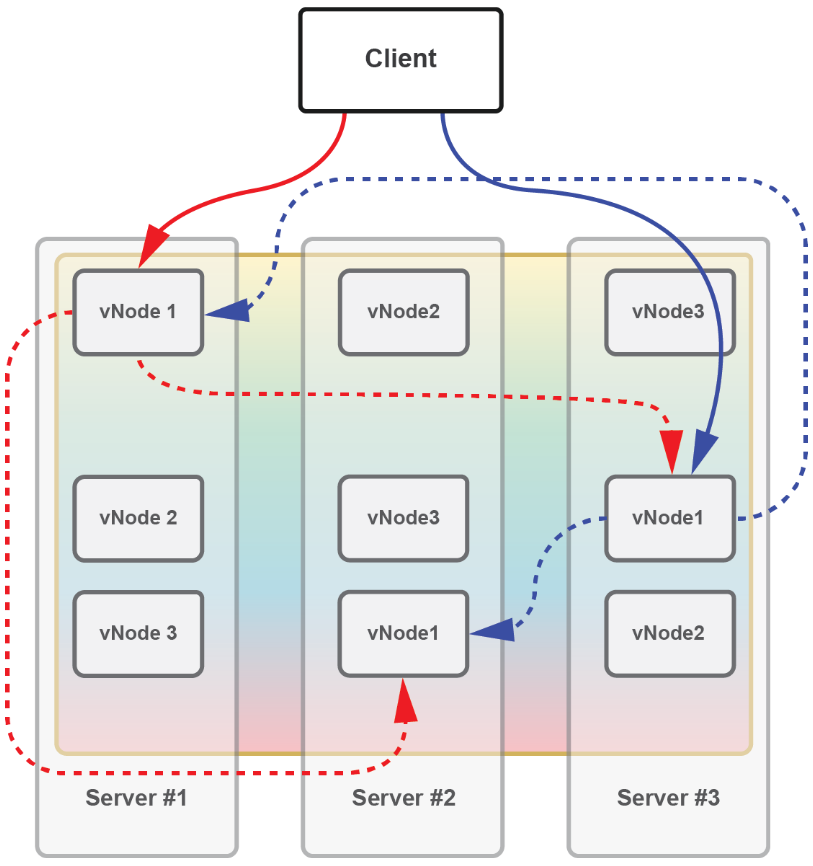


Image 3.12. The intra-cluster replication in Cassandra

The multi-cluster replication can be configured using an appropriate replication strategy and a *snitch*, which implements node data center awareness and rack awareness. The replication strategy distributes replicas based on the information provided by the snitch. There are several tunable consistency levels for a multi-cluster deployment. It is possible to either acknowledge a write request in a single data center and asynchronously replicate it to other data centers or wait for the responses from multiple data centers before the final acknowledgement.

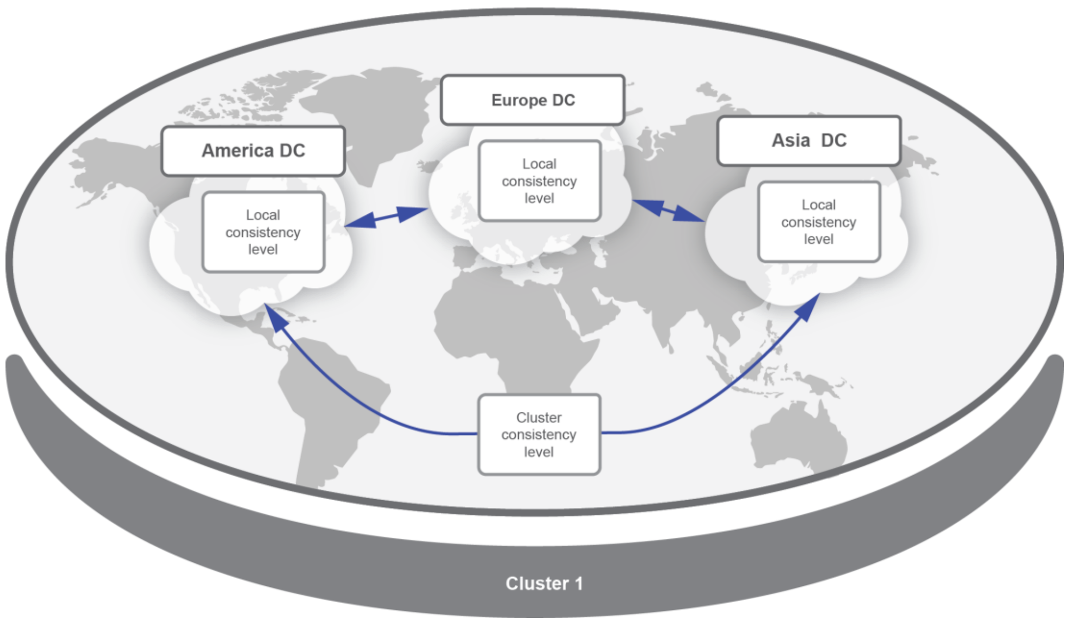


Image 3.13. The multi-datacenter replication in Cassandra

In addition, the advanced replication provided by DataStax allows you to build such topologies as the hub-and-spoke or mesh networks, identifying source and destination clusters with the replication channels.

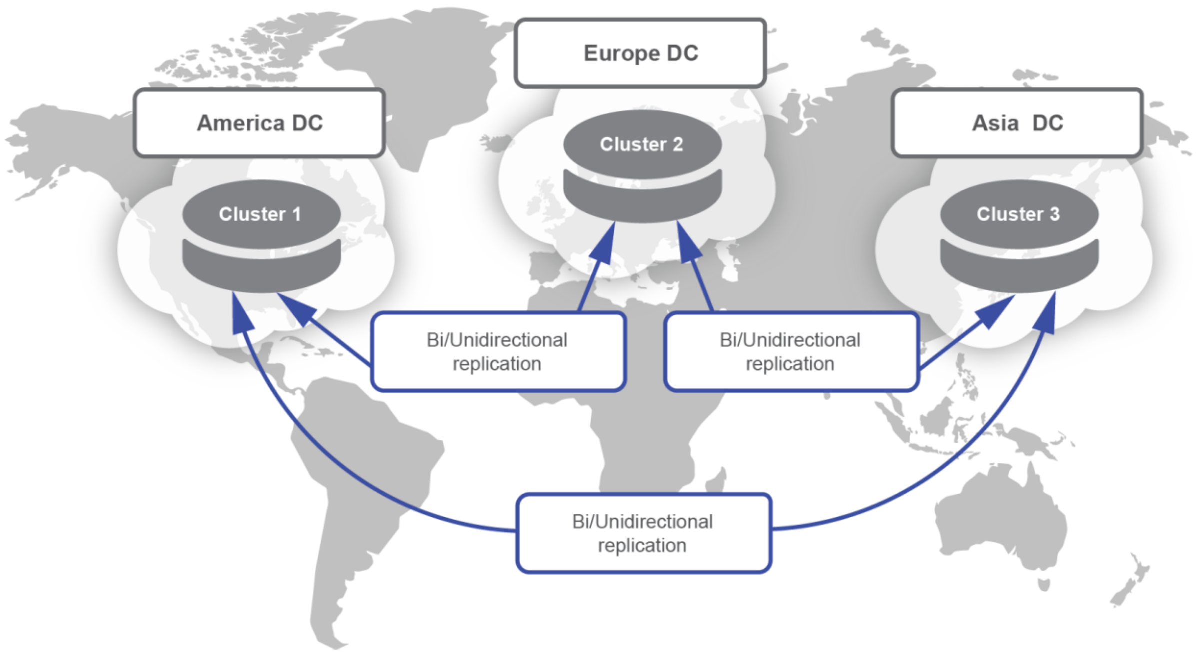

Europe DC
America DC
Asia DC
Cluster 1
Cluster 2
Cluster 3
Bi/Unidirectional replication
Bi/Unidirectional replication
Bi/Unidirectional replication

Image 3.14. The multi-cluster replication by DataStax

MongoDB

In MongoDB, cluster data is horizontally partitioned across independent groups of the mongod processes. Each group, called a replica set, maintains redundant copies of the same data partition through asynchronous replication within the group. Keeping copies of data on multiple database servers allows the system to provide fault tolerance against the loss of a single cluster node. In some cases, it helps to achieve increased read capacity by enabling replica reads, though, at a cost of consistency. Maintaining data replicas in different data centers increases data locality and availability for geographically distributed applications. However, the proper replica set election and sharding process must be designed and configured for such multi-datacenter deployments.

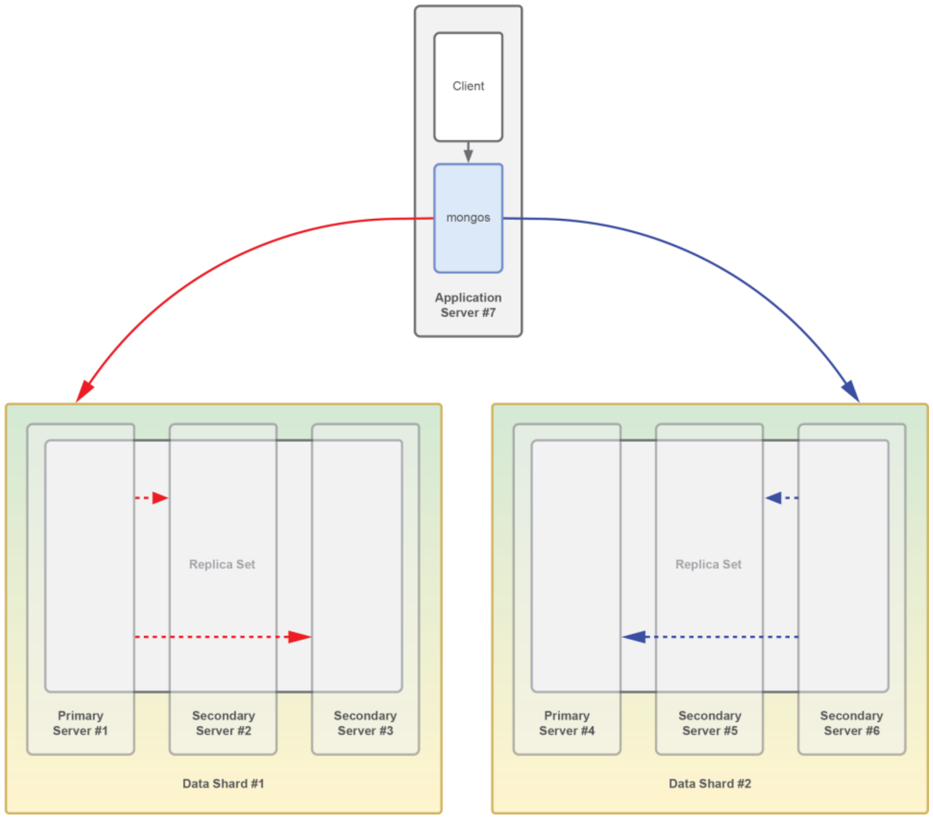


Image 3.15. The intra-cluster replication in MongoDB

A replica set has an exclusive node, called primary, that mandatorily receives all the write operations. It records all its data mutations to an operation log (also called an “oplog”). Secondary members asynchronously replicate the primary's data by reading its oplog and applying the operations to their own data. Thus, within a replica set, each secondary’s data reflects the primary’s data state at some point in time. Each record in the oplog is idempotent—i.e., the oplog operation produces the same result whether applied once or multiple times to the target dataset. The oplog itself is a circular buffer of a configurable size implemented as a special MongoDB capped (fixed-size) collection.

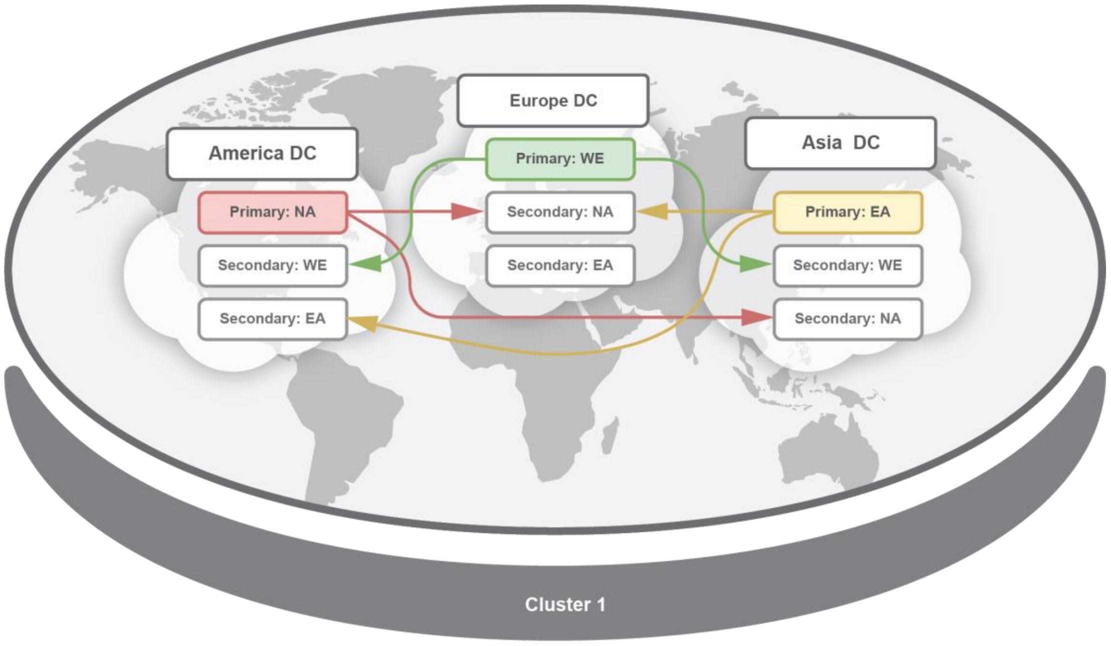


Image 3.16. The multi-datacenter replication (read local/write global) in MongoDB

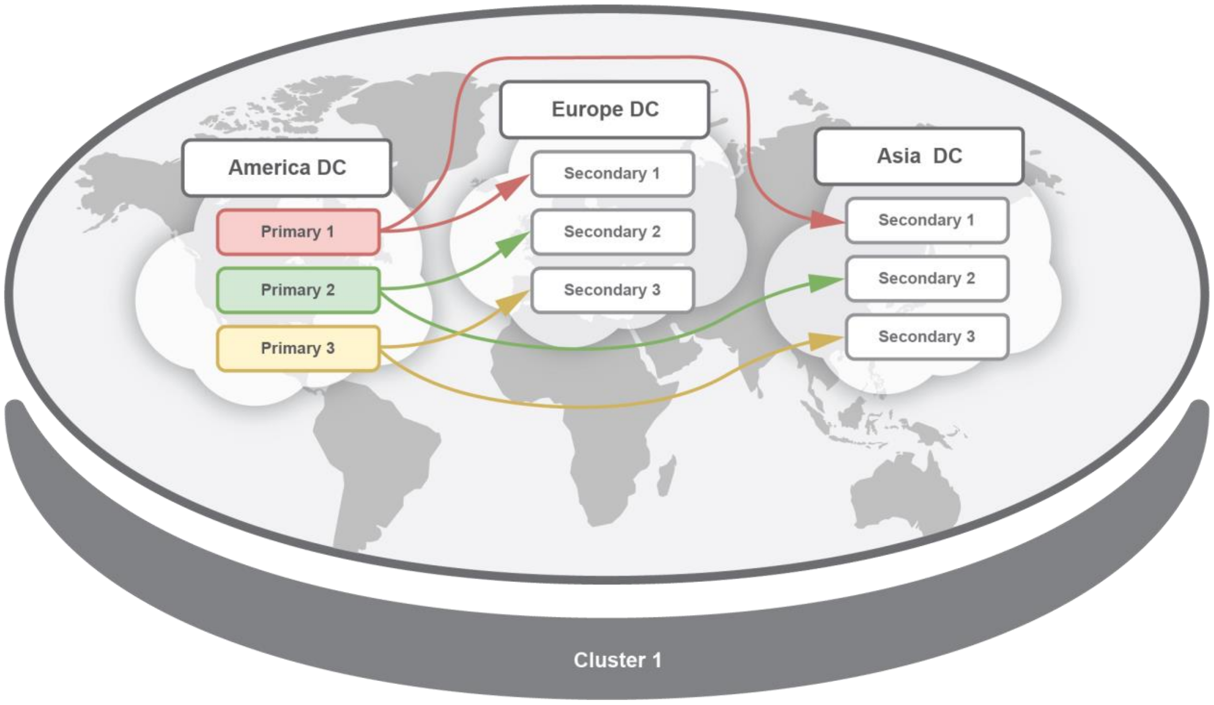


Image 3.17. The multi-datacenter replication (active-standby) in MongoDB

Summary

All the three compared solutions feature sophisticated partitioning and replication mechanisms. Each solution has a fault-tolerant design both for single- and multi-datacenter deployments with high uptime. The MongoDB implementation of partitioning and replication is decoupled—it gives additional flexibility at a cost of the increased complexity. However, MongoDB does not support and thus does not allow to configure advanced multi-cluster deployments.

Table 3.5. Replication of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
9	10	10

3.1.6 Server and Network Fault Tolerance

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

Couchbase Server

In terms of the CAP theorem, Couchbase Server is CP—it provides consistency and partition tolerance. Data is horizontally partitioned, replicated, and distributed across the cluster servers. Each data partition, called a vBucket, has an active copy and up to three replica copies, which are intelligently distributed across the server nodes and racks. If a node mastering an active vBucket fails, it is possible to send read requests to replicas. The process that recovers from an active vBucket failure is called a "failover." It promotes replica vBuckets, residing on different nodes, to become active vBuckets. The failover can be either manual or automatic.

In case of a network partition, the automatic failover is triggered only if the following conditions are satisfied:

- A single node is down at a time.
- There has not occurred a non-verified automatic failover before.
- The cluster has been observing communication issues for at least ten seconds.

The minimum delay is necessary to prevent the transient network issues or a temporary node freeze from triggering the failover.

So, if there is a network partition between several nodes in a cluster, the automatic failover is not triggered. However, the remaining sub-clusters continue to operate properly and provide strong data consistency, since each active vBucket is responsible for its own subset of keys and they do not intersect. The cluster will remain in this state until the partition is resolved or an administrator intervenes. If the partition is resolved, Couchbase will automatically resume any replication that was blocked, and no manual repair or action is needed.

In contrast to the Data Service, the Full Text Search, Query, and Index services can be failed over without any recovery delay because they are ready-only and can be load-balanced more gracefully. A Cluster Manager provides load balancing out of the box.

Cassandra

Partition tolerance is one of the major architecture principles of Cassandra. A peer-to-peer network topology and tunable replication make Cassandra strongly fault-tolerant with no single point of failure. A specified number of replicas are stored on different nodes in a single-datacenter or a multi-datacenter cluster. All the replicas have the same role, so their failures do not affect each other. If a node goes down, the whole system continues to work with the reduced throughput. The missing write requests will be replayed on the failed node after it rejoins the cluster by a special process called a hinted handoff. This process involves a replica storing incoming requests—while the other replica is offline—and then sharing these requests with the second replica.

If a split brain occurs due to the network partition, the resulting sub-clusters independently continue to process the incoming requests without any delay. After the network connection between sub-clusters is fixed, the hinted handoff process is used to exchange the missing write requests.

MongoDB

In terms of the CAP theorem, MongoDB is focused on consistency and partition tolerance. It employs the master-slave architecture within a replica set both for data shards and config servers. The automatic failover, which is a default behavior, implies that the replica set's secondary becomes the new primary if the old primary is unavailable. While the elections are in progress, the replica set has no primary and cannot accept write requests.

The network partition can split out the replica set's primary. If the primary detects that it cannot see the majority of the nodes in the replica set, the primary becomes a secondary.

Data shards and config servers must be deployed as replica sets tolerate cluster node failures. However, if the config server replica set loses its primary and for any reason cannot elect a new one, the whole cluster enters the read-only mode. If all the config servers or mongoses are unavailable, the cluster becomes inoperable.

Summary

Cassandra tolerates network partition or multiple server failures without any significant timeout. Couchbase and MongoDB require a certain amount of time for recovery. If MongoDB loses all configuration servers or router processes, the whole system becomes unavailable.

Table 3.6. Fault tolerance of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
7	9	10

3.2 Administration

3.2.1 Recovery

Recovery includes the policies and procedures required to get databases up and running after single-node failures.

Couchbase Server

In the event of a server failure, the failover process must be triggered, either automatically by a cluster or manually by an administrator/script. The failover operation immediately removes the node from the cluster and activates the replicas for its data that are spread across the rest of the cluster. There is no data to be moved as these replicas have been kept up-to-date, so the failover itself is immediate. Once the failover process happens, some vBuckets are missing a replica copy. At this point, a manual rebalance operation is required either to recreate the replicas within the remaining nodes (at the administrator's discretion) or the node should be replaced and a rebalance performed. Either way, a rebalance is needed to return the cluster to a healthy and fully replicated state. The rebalance happens online with no impact to the application's configuration, performance, or availability.

If the failed-over node becomes available again, it is possible to return it back to the cluster using either the full or delta recovery method. The full recovery strategy completely cleans up the failed-over node and restores data from other cluster nodes. The delta recovery strategy reuses the data on the failed-over node. It checks the existing data to identify the point when mutations stopped, and synchronization begins at that point. As only delta changes are being restored, the network resource usage is minimized, and, therefore, the recovery time is significantly faster. However, the delta node recovery is available only if no rebalance operation occurred before, since the rebalance operation changes the cluster topology and the vBuckets map. In that case, the delta recovery strategy defaults to a full recovery.

Cassandra

To replace a failed node in Cassandra, you should start the replacing node with a specific property pointing to the IP address of the failed node. The replacing node starts bootstrapping data for the same token ranges from the rest of the nodes in the cluster. While bootstrapping is in progress, the new node does not accept any writes and seems to be down for other nodes. The hinted handoff is used to resolve inconsistency, but if a node is down for too long, the manual read repair process is required.

Another option is to remove the failed node using the *nodetool removenode* command and then add a new node to the cluster. When the failed node gets removed from the cluster, its token ranges are assigned and transferred to other nodes. The data for the new node can be restored from a backup or bootstrapped from other nodes. Restoring from a backup allows you to restore data without copying it from other nodes, but it also requires read repair afterwards to resolve inconsistency. In contrast to the backup approach, bootstrapping preserves consistency but causes additional data streaming from other nodes.

In comparison to the direct replacement and restore from a backup, bootstrapping is less effective because it rebalances cluster data twice.

MongoDB

The MongoDB replica set members use a continuous heartbeat message exchange to detect a node failure. The node with a missing heartbeat will be marked by other members as inaccessible. If the missing node was the replica set primary member, an eligible secondary holds elections to promote itself to become a new primary. Elections take time to complete, and while they are in progress, the replica set has no primary and cannot accept writes. All the other members stay in the read-only mode, as well. After the elections are finished, the replica set continues its normal functioning, but with reduced redundancy. It also may or may not reduce the replica set read capacity depending on the used read preference.

As a replica set member is fixed and rejoins, it may either copy the oplog from its sync source to apply the operations or perform a full copy. However, a full copy is only initiated if the rejoining node has become too stale during its unavailability. (Due to the fact that the oplog is a circular buffer and the last operation the rejoining node knows about has been overwritten on the sync source, the replication cannot be completed.) The oplog size is configurable, and the sync source is automatically selected based on the heartbeat (ping) results.

A MongoDB user might suffer data loss as a result of a “roll back.” You may choose the write concern option that specifies how many replica set members the primary will wait for before returning the acknowledgement to the issuing client. The default write concern is one, but you must use the *majority* write concern so as not to lose updates when the former primary member rejoins its replica set after a failover. As a result of the rejoin, it might become a secondary with a diverged oplog. This divergence is fixed by the roll back procedure that discards the updates acknowledged by the former primary, still never being replicated to the secondaries.

MongoDB can also be recovered from a backup. The Ops/Cloud Manager enables the point-in-time restoration for a single MongoDB process, replica set, or the entire sharded cluster. To make this type of recovery available, you configure the snapshot frequency and retention policy, enable checkpoints, and specify a time interval between those checkpoints.

Summary

When all the solutions are properly configured, with all the potential risks of data loss taken into account, they provide durable recovery for single-node failure accidents. The implementation of an effective recovery strategy for the MongoDB sharded deployment requires a solid understanding of the database architecture.

Table 3.7. Recovery of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
9	10	10

3.2.2 Disaster Recovery

Disaster recovery includes policies and procedures that enable database recovery after global system disasters.

Couchbase server

Couchbase Server provides several ways to recover a cluster after a global system disaster. The first one is to enable periodic incremental backups using either the cbbackupmgr tool or the cbbackup and cbrestore tools. If the periodic backup is set up and running, then in case of a serious failure, you will be able to restore data from backups. However, the snapshot of the entire cluster might be a bit outdated because it is impossible to create a complete in-time backup.

The other way to recover a cluster is by means of XDCR. If a cluster suffers a failure and requires a partition recovery, it is possible to recover the missing data partition from the backup cluster using the cbrecovery tool. This tool allows to preview a list of buckets that are no longer available in the cluster and recover the missing buckets from the remote cluster.

Cassandra

One of the ways to handle a global disaster is to recover a cluster with the help of backups. Cassandra provides snapshots, as well as incremental and commit log backups. Any of these types of backups are suitable for recovery, but each has a different degree of data consistency and relevance.

Another technique for disaster recovery is a multi-datacenter replication. A single data center handles client requests and asynchronously replicates them to the other one, which serves as a live backup that can quickly be used as a fallback cluster. After the original data center returns back online, the manual repair process should be run on all of its nodes using the nodetool command-line utility or OpsCenter Web UI.

MongoDB

MongoDB offers two major disaster recovery strategies. One leverages continuous backups, while the other makes use of multi-datacenter deployments.

A continuous Ops/Cloud Manager backup is an ongoing process that works in the similar fashion as the replica set data synchronization. By enabling checkpoints, you can permit point-in-time restores in between the snapshots are taken.

A typical disaster recovery strategy involves keeping each shard replica set member distributed across two data centers: an active and a standby. All the primaries and first-order secondaries live in the active data center, and at least one secondary for each data shard is kept in the standby data center. The elections are configured in the same way—the backup secondaries are promoted to become primaries. However, this is done manually in case the active data center disaster is confirmed.

Summary

All the three solutions provide the necessary tooling to perform a graceful recovery after a global system disaster. They enable data recovery either from backups or by means of a remote redundant cluster.

Table 3.8. Disaster recovery of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	10

3.2.3 Backup

Backup refers to the capabilities and convenience of copying and archiving cluster data, so it may be used to restore the original database state after data loss or a corruption event occurred.

Couchbase Server

Couchbase Server comes with the cbbackupmgr (Couchbase Backup Manager) tool. It combines backup and restore facilities previously held by the cbbackup (Couchbase backup) and cbrestore (Couchbase restore) tools. It performs the first full backup followed by incremental backups. It backs up and restores bucket data, view and index creation scripts, and bucket configurations.

Couchbase Server enables you to perform two types of an incremental backup: differential and cumulative ones. A differential backup contains only the changes that have occurred since the previous backup. A cumulative backup includes all the changes made since the last full backup.

You can employ a variety of backup strategies. The *periodic merge* strategy creates and later merges incremental backups after a specified period of time using a single backup repository. The *periodic full plus incremental* approach creates a fresh full backup in a new repository instead of merging incremental backups. The *full backup only* strategy creates a new full backup each time. Additionally, unidirectional XDCR can be used as a remote cluster data backup.

Due to the asynchronous and distributed nature of Couchbase Server, it is not possible to create a complete point-in-time backup of the entire cluster since data is being continuously updated.

Cassandra

The immutability of the SSTable files allows Cassandra to make snapshots by hard-linking the original files instead of copying. After the snapshot is complete, these files should be copied to another location. You can use the nodetool to make and remove a snapshot on a particular node. Making a snapshot of the entire cluster requires advanced tools, such as OpsCenter. Cassandra also provides a configurable “auto snapshot” feature that backs up a keyspace or a table before it gets dropped or truncated.

In addition, Cassandra supports incremental backups that continuously hard-link the newly flushed SSTables. The commit log backups facilitate the SSTable backup data for the later recovery to a particular point-in-time state. Cassandra does not provide a utility for automatic scheduling and removing backups, so it should be done manually.

MongoDB

MongoDB supports several methods to enable backups and restore data. Its Cloud/Ops Manager continuously backs up a cluster by reading the oplog data. It creates snapshots of the cluster data at specified intervals and can also offer the point-in-time recovery (as the oplog contains the operations’ timestamps and checkpoint tokens).

The Cloud/Ops Manager is the preferable way to perform cluster backups. Still, you can also initiate a MongoDB cluster backup by copying the underlying data files (the file system snapshots). This method is provided by the operating system volume manager and is not specific to MongoDB, but is in common use. The file system snapshots are created quickly and they work reliably, though you have to make more configuration changes outside of MongoDB. The MongoDB backup and restore utilities (mongodump and mongorestore) work well only for the small-scale deployments.

Summary

All the three products provide a rich set of backup and restore tools, and each product has its own pros and cons. Unlike Cassandra and MongoDB, Couchbase Server is not able to create the point-in-time backups. However, Couchbase does allow you to back up data on a remote cluster by the means of XDCR. Cassandra requires significant effort to automate the backup process.

Table 3.9. Backup capabilities and convenience of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
9	9	9

3.2.4 Configuration management

Configuration management embodies the means of keeping cluster nodes configuration in a coherent state during both system bootstrap and runtime. Ease of configuration management can dramatically improve user experience with a product.

Couchbase Server

A Couchbase Server cluster can be configured from an arbitrary node using the Web UI or CLI, both based upon the supported REST API. Multidimensional scaling—independent scaling of the Data, Query, and Index services—can be done through the Web Console UI just by turning the corresponding services on and off while adding a new node to a cluster. Alternatively, you can conduct multidimensional scaling via the CLI and the REST API. The intra-cluster and cross-region replications are configured per bucket. Administrative tasks can be performed and automated using the REST API.

Cassandra

Several property files are used for the Cassandra cluster configuration. They are responsible for configuring a network, security, main memory and persistent storage, logging, backups, compression, etc. The nodetool is able to update the configuration properties in a runtime for a single node only. OpsCenter allows you to configure and apply changes to a particular node or to the whole cluster using the Web UI. DataStax Search and Analytics also can be managed using OpsCenter. Due to the strong cohesion between the number of replicas and consistency levels in Cassandra, customer applications must be aware of the database deployment details (a replication factor in particular) to operate properly on all the development environments.

MongoDB

The bootstrapping properties for a MongoDB cluster can be specified with the YAML configuration files or the CLI. The MongoDB shell is used to manage cluster configuration in a runtime. Additionally, MongoDB Cloud Manager (a hosted service) and Ops Manager (an on-premises solution) provide cluster deployment automation, configuration management, monitoring, and other capabilities.

Summary

All the three products provide a Web UI to configure particular nodes or the whole cluster.

Table 3.10. Ease of configuration management of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
9	10	9

3.2.5 Maintenance

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

Couchbase Server

You can maintain Couchbase Server through the Couchbase Web Console, the CLI, and the REST API. The Couchbase Web Console tool is the most convenient one and provides a complete interface for configuring, managing, and monitoring a cluster except for backup and restore capabilities, which are facilitated by the CLI tools.

Couchbase Server is designed to perform maintenance tasks online. The rebalance operation allows you to add and remove one or several Data Service nodes without any downtime. The change in the Query and Index Services topology requires no data movement, so one can add or remove nodes on the fly. The cluster map changes are automatically broadcast to SDKs that begin load-balancing across those services. Couchbase Server is also capable of handling any hardware or database configuration changes online. A single node failure can be detected and handled by the automatic failover. However, later on it will require a manual rebalance operation trigger. In addition, a multiple node failure can be resolved only by a manual intervention.

Cassandra

All the required maintenance tasks can be performed through the nodetool command-line utility, which is embedded in the Apache Cassandra distribution or available through the DataStax OpsCenter Web UI. The nodetool runs commands for a particular cluster node, while OpsCenter enables you to execute tasks for all the nodes in the cluster. Typical maintenance tasks, such as updating the OS or Cassandra version, are performed online without any downtime. Vnodes are evenly redistributed among all the cluster nodes during such activities.

One of the important maintenance tasks that you should perform consistently on a Cassandra cluster is the anti-entropy repair. Its goal is to resolve inconsistency between replicas to guarantee eventual consistency. This process runs in the background and can cause the degradation of requests latency and throughput performance. The continuous usage of incremental and range repairs helps to minimize such an impact.

MongoDB

All the maintenance tasks for the cluster can be performed in MongoDB Ops/Cloud Manager, which is part of the MongoDB Enterprise Advanced framework. The cluster management solution requires installation of various agent types on server nodes, including the automation, backup, and monitoring agents. They introduce relatively slight overhead on the cluster hardware requirements, thereby providing a user with a comprehensive set of the cluster maintenance capabilities.

The MongoDB Ops Manager enables scaling in and out without taking the application offline, performing topology changes, upgrading without any downtime, and performing the point-in-time scheduled backups.

Summary

Couchbase Server ships with a Web Console that is designed to perform the majority of maintenance tasks in a single place. Cassandra and MongoDB do not provide such a tool out of the box, but have similar solutions that must be set up separately and require the allocation of additional resources. In case of MongoDB, the Ops/Cloud Manager installation also requires a manual setup of different agents.

Major maintenance activities, such as adding or removing nodes, or vertical scaling, are handled without any downtime. However, in case of Cassandra, any topology changes can cause performance degradation.

Table 3.11. Maintenance of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
8	10	9

3.2.6 Monitoring

Monitoring involves all the efforts required to examine the resources used, check on performance, and gather statistics.

Couchbase Server

Couchbase Server comes with several different monitoring capabilities, including the Web Console, the REST API, and the cbstats utility.

Web Console provides a centralized view of all cluster metrics across different categories—hardware resources, data, replication, the XDCR timestamp-based conflict resolution, etc. Additionally, it displays visual notifications in case of any accidents and supplies a configurable alerting system via e-mail for any significant errors related to the cluster.

The Couchbase REST API exposes endpoints, providing cluster-, node-, and bucket-level statistics.

Couchbase Server stores per-node statistics that can be extracted using a client—implementing the binary protocol—or with the help of the cbstats utility.

Cassandra

Cassandra exposes a number of metrics and operations via the Java Management Extensions (JMX). JMX is a Java technology that supplies tools for monitoring and managing Java applications. It is possible to view and analyze Cassandra metrics using several JMX compliant tools, including the nodetool, OpsCenter, and JConsole. The nodetool command-line interface, which is embedded in Cassandra distribution, facilitates monitoring and maintenance. OpsCenter is a graphical web UI that allows you to monitor all the nodes in a cluster from a single place. JConsole is a built-in Java graphical desktop user interface that consumes and displays the JMX metrics. It is also possible to poll Cassandra metrics through the JMX HTTP bridge.

MongoDB

There are several complementary methods for collecting data about the state of a running MongoDB cluster. A set of utilities distributed with MongoDB provides real-time reporting of the database activities. Mongostat captures and returns the counts of database operations by type (such as insert, query, update, delete, etc.) and per server. Mongotop tracks and reports the current read and write activity of a MongoDB instance, and reports these statistics on a per-collection basis.

The MongoDB shell commands return statistics regarding the target database state with greater fidelity. Accessible information includes details on disk and memory usage, connections, journaling, index access per database or collection, a replica set state and configuration, and the statistics about replica set members. MongoDB Ops and Cloud Manager provide comprehensive monitoring (by installing light-weight monitoring agents on the cluster nodes), visualization, and an alerting system. Various third-party monitoring tools, such as Ganglia, Nagios, Munin, and Wireshark, support MongoDB, either directly or through their plugins.

Summary

All the solutions under consideration allow you to perform monitoring using the command line shell and Web UI. In addition, Couchbase Server and Cassandra provide a possibility to access the database statistics through the REST API.

Table 3.12. Ease of monitoring NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
9	10	10

3.2.7 Security

Security embraces authentication, access control, data store, and transfer encryption.

Couchbase Server

Couchbase Server offers internal credentials-based authentication. The enterprise edition, when running on a Linux machine, additionally allows you to implement authentication by means of the Lightweight Directory Access Protocol (LDAP) or Pluggable Authentication Modules (PAMs).

For authorization, Couchbase Server comes with RBAC. It provides built-in roles that can be assigned per user or per bucket for each data access type (such as key-value, N1QL, and FTS). Each role contains a set of privileges, such as read, write, list, execute, etc.

Data encryption can be achieved both at-rest and over-the-wire. To encrypt data at-rest (on physical media), Couchbase Server supports transparent data encryption tools including EFS on Windows, Linux Unified Key Setup (LUKS), encrypted EBS volumes (in AWS), Vormetric, Gemalto, and Protegrity.

Over-the-wire, client-server, as well as XDCR traffic encryption are achieved using TLS (versions 1.0–1.2) via either the self-signed or managed X.509 certificates. The intra-cluster replication is unencrypted out-of-the-box, and Couchbase recommends to configure the Internet Protocol Security (IPsec) framework. However, it should be configured manually on each node.

All the REST/web interfaces support HTTPS. All the passwords and internal tokens are hashed when stored on a disk.

Couchbase Server performs auditing and captures information about a user, an action taken, a date, and the results. Audit logs can be retrieved and inspected later by a system administrator.

Cassandra

Cassandra uses a concept of roles for authentication and authorization. Each role represents a single user or a group of users. Both authentication and authorization are pluggable and disabled by default. Username/password authentication stores encrypted credentials in system tables. DataStax additionally supports authentication using LDAP and Kerberos. The authorization solution embedded in Cassandra features the permissions management functionality and stores its data internally. Using LDAP, you can also manage permissions on DataStax.

Cassandra secures both the client-cluster and intra-cluster communications using the TLS/SSL encryption. These encryption types are managed separately and may be configured independently. Transparent data encryption provided by DataStax allows you to encrypt such data at-rest as tables, indexes, commit logs, and property files. Auditing provided by DataStax enables system access control.

MongoDB

MongoDB supports the following authentication mechanisms: SCRAM-SHA-1, x.509, LDAP, and Kerberos. RBAC is available. A role grants privileges to perform the specified actions on the specified resource. Each privilege is either specified explicitly in a role, or inherited from another role, or both. A resource is a database, a collection, a set of collections, or a cluster. Actions are queries and writes, database management operations, replication or sharding setup procedures, etc. There also exists a set of the predefined built-in roles.

MongoDB provides administrators with a system auditing facility that can record system events, such as user operations and connection events. These audit records permit forensic analysis and allow administrators to verify proper controls over the system access rights.

To encrypt communication between cluster components, MongoDB uses TLS/SSL for all incoming and outgoing connections. The WiredTiger storage engine provides encryption at rest that can be configured to encrypt data in the storage layer.

MongoDB, Inc. provides the Security Technical Implementation Guide (STIG) on request. It contains security guidelines for deployments within the United States Department of Defense. For applications requiring the HIPAA or PCI-DSS compliance, users can refer to the publicly available MongoDB Security Reference Architecture.

Summary

All the three solutions provide RBAC, include the ability to integrate major authentication and authorization mechanisms, such as LDAP, allow you to encrypt data at rest, enable you to secure intra-cluster and cross-datacenter traffic, and perform audits. In addition, MongoDB supports STIG.

Table 3.13. Security of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
9	9	9

3.3 Development

3.3.1 Data structure and format

The storage and index data structures are directly responsible for space amplification and efficiency of the read and write operations. The data format defines the way components in a single data record are treated—for example, a row, a document, a key-value pair, etc.

Couchbase Server

Couchbase Server stores data records (documents) organized in buckets. Each record consists of a unique user-defined key, a value, and metadata. The key must conform to the UTF-8 encoding and be no longer than 250 bytes. The value of a stored record can be a JSON document or any binary data. Buckets can mix different record types. They do not require a predefined schema.

Different storage engines are used depending on a selected bucket type. Currently, there are three bucket types available: Couchbase, Ephemeral, and Memcached. Both Ephemeral and Memcached bucket types are in-memory only. The Couchbase and Ephemeral bucket types support replication, indexing, backup, full-text search, etc. The Ephemeral buckets provide higher performance compared to the Couchbase buckets, but at a cost of persistence. The Couchbase bucket persistence is provided by the on-disk B+Tree-based engine called Couchstore. The Memcached bucket type is provided primarily for the Memcached protocol compatibility.

Couchbase Server provides a number of means to enable querying for documents by attributes—Incremental MapReduce Views, Spatial Views, Full-Text Search, and GSIs. Views extract specific attributes, aggregates, and any other type of information from documents by the user-defined MapReduce functions for the querying purposes. Global Secondary Indexes are used to retrieve data via the SQL-like syntax. The GSIs have two storage modes: standard and in-memory. The in-memory mode is implemented through a lock-free skiplist and requires enough RAM to fit the entire index. If RAM is exhausted, the index will stop processing new updates but is still available to serve queries. The standard mode leverages the same lock-free skiplist in memory but also persists to blocks on a disk. It allows an index to be larger than the RAM available at the potential cost of performance via disk access.

The Full-Text Search indexes are implemented with a structure similar to a log-structured merge (LSM) tree on a disk—a stack of the sorted, immutable key-value arrays that are then dynamically mapped into memory for increased performance.

Cassandra

Cassandra is a partitioned row store. Rows consist of columns and are organized into tables, and tables are grouped in keyspaces. A unique primary key is required for each table. The key consists of a mandatory *partition* key and an optional *clustering* key. The partition key is used for breaking table data into partitions, which are distributed across different nodes in a cluster. The clustering key is responsible for sorting and grouping data within the partition. Data can be queried and inserted in the JSON format, but the insertable JSON has to reflect the table schema. DataStax Enterprise also provides support for graphs. A proper table structure, partition, and clustering keys must be worked out by a user.

To query by columns, which are not part of a primary key, Cassandra uses *secondary indexes* and *materialized views*. Secondary indexes and materialized views are implemented as regular tables with their own primary keys, which are kept in sync with the original tables. Secondary indexes are processed locally. Each index partition is stored on the same node as its original data partition. In contrast, materialized views are partitioned independently from the original tables. In general, secondary indexes are built faster during writes, but they are less efficient during reads than materialized views.

When a write occurs, it is first stored in a memory data structure called a *memtable* and also appended to a *commit log* on a disk for the recovery purpose. When the memtable exceeds the configurable capacity, the threshold data is flushed on a disk to an SSTable and the commit log for this portion of data is purged. SSTables and memtables are maintained per table. SSTables are immutable and cannot be modified after a memtable is flushed. An LSM tree is the underlying data structure behind a SSTable.

MongoDB

MongoDB structures data into collections of the Binary JSON (BSON) documents. BSON is a binary-encoded serialization of the JSON-like documents. Similar to JSON, BSON supports embedding documents and arrays within other documents and arrays.

You can configure MongoDB to use one of the following storage engines on a per-node basis: WiredTiger (default), in-memory, and MMAPv1. WiredTiger is a multi-purpose back end that uses a write-ahead transaction log in combination with checkpoints (a snapshot of the data state on a disk) and provides the document-level concurrency model. WiredTiger supports both the LSM tree and B-tree table types. However, only the B-tree table types are currently used within MongoDB. WiredTiger is also used as an in-memory storage engine back end. MMAPv1 is a legacy storage engine.

A MongoDB cluster allows for mixing storage engines within its replica sets in any combination. For example, you can use the in-memory storage engine for primaries to achieve better latency and throughput, and the WiredTiger storage engine for secondaries to improve durability.

MongoDB supports various index types, including single field, compound, multikey, text, geospatial, etc. In a sharded cluster, indexes are built on the replica set primaries first and replicated to the secondaries on completion.

Summary

The effectiveness of a NoSQL solution depends on the suitability of the underlying architectural concepts for each particular workload. All the three databases leverage adequate algorithms to efficiently read and write, and utilize disk space and memory in accordance with the particular data structures and formats. Couchbase Server and MongoDB are appropriate solutions for storing the unstructured JSON documents that should be queryable across multiple fields, as they provide JSON support out of the box. Cassandra is a good choice for storing well-structured data and requires you to determine how to best store, manage, and access it. Cassandra requires preliminary data modeling effort, which may result in more efficient data querying.

Table 3.14. Data structure and format of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	10

3.3.2 A query language

A query language is a type of a programming language that modifies and retrieves data from a database by sending queries.

Couchbase Server

Querying by a document key is the simplest and most efficient way of accessing the Couchbase Server data. The key-value request is sent directly to a proper node holding the target document and does not require any index scanning. The Couchbase Server views enable indexing and querying data by document attributes. This way of accessing data requires a user to create the MapReduce functions that are incrementally applied to each incoming mutation. Querying data with spatial views enables multidimensional bounding box queries for the location-aware applications.

N1QL is a Couchbase SQL-like query language with a set of specific extensions for addressing JSON data. It features data query, data manipulation, and data definition capabilities, as well as role-based access control (RBAC). N1QL facilitates a sophisticated JOIN capability to query over multiple documents and other SQL-like features. In addition, N1QL provides tools to manage query performance as prepared statements, allowing to skip the parsing and execution planning phases for repeating queries. N1QL also offers the explain plan option to investigate query execution flow and timings.

As with SQL, it is possible to use basic CRUD operations, nested subqueries, joining, filtering, grouping, ordering, and paging. N1QL also features a rich set of functions to transform the query result. A user can improve query performance with secondary indexes and implement covered index queries.

Cassandra

Cassandra Query Language (CQL) is a SQL-like language for data manipulation, definition, and control. In addition to the core SQL data types, CQL supports several collection types (a list, a set, and a map) and allows for creating user-defined types. The language provides CRUD operations for data manipulation. Filtering, ordering, and grouping queries are also supported but must be constructed in accordance with a particular table structure. JOIN clauses are not supported. Keyspaces, tables, indexes, and triggers can be defined, altered, and dropped using CQL. RBAC is also managed through the query language commands. A set of scalar and aggregation functions are provided out of the box. You can also define specific user functions with Java virtual machine-compliant (JVM) programming languages.

MongoDB

MongoDB provides a user with its own API-based query language, comprising CRUD operations, an aggregation pipeline, and MapReduce.

MongoDB CRUD operations can be applied to a single document or multiple documents. However, an operation is atomic only at the level of a single document. When a single write operation modifies multiple documents, the modification of each document is atomic, but the operation as a whole is not atomic and other operations may interleave. Still, you can isolate a single write operation that affects multiple documents using a special isolated operator.

The aggregation pipeline is a framework leveraging the concept of the data processing pipelines. Documents pass through a multi-stage pipeline that transforms them into the aggregated results. Each pipeline stage may either produce zero, one, or multiple output documents per input document. Some stages take a pipeline expression as an operand that specifies the document transformation. MongoDB provides a large set of stages and expressions out of the box that cover most of the SQL functionality.

The functionally similar to the aggregation pipeline, MapReduce transfers a collection of documents through two stages—map and reduce—that project and group the documents. Both stages are defined in JavaScript. In comparison to the aggregation pipeline, MapReduce provides more flexibility at a cost of the decreased performance and increased complexity.

Summary

Couchbase Server and Cassandra offer users the SQL-like query languages of their own. However, N1QL is more flexible and provides more SQL features comparing to CQL. MongoDB provides a non-standard query API that covers most of the SQL features, but requires a developer to learn the specific coding idiosyncrasies. CQL allows to perform filtering, ordering, and grouping operations only in a strict accordance with the table structure. The N1QL and MongoDB query languages do not have such a restriction, but on the other hand, require indexes creation to achieve similar efficiency. Both MongoDB and Couchbase Server implement the MapReduce paradigm for sophisticated user-defined data transformations.

Table 3.15. The query language capabilities of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
8	10	8

3.3.3 Full-text search

Full-text search refers to the capability of a database to perform an effective search against textual information in any stored document.

Couchbase Server

Full Text Search (FTS) Service is an integration of the text indexing and a the Bleve search library in Couchbase Server. It allows you to perform full-text search across specified document attributes or the entire document content. FTS provides a text analysis capability and comes with a number of pre-built analyzers out of the box. These analyzers make a language-aware search possible and offer support for terms, numeric, and date-range facets. Search results are ordered by relevance using the TF-IDF scoring.

To perform a full-text search, you need to create a full-text index for a selected bucket. The full-text indexes are automatically partitioned, distributed, and replicated across the Couchbase Search Service nodes. If one of the Search Service nodes fails, FTS returns a list of partial results and notifies about the error.

Cassandra

Cassandra does not support full-text search out of the box, but DataStax Enterprise Search provides this ability by integrating Cassandra with Apache Lucene and Apache Solr. Lucene is a search engine framework that adds search capabilities to an application via the Java API. Solr is a standalone web application that uses Lucene as its core and ships additional enterprise features. Lucene and Solr are closely integrated and run in a single JVM.

Data is asynchronously indexed in Solr during the Cassandra write path. The ad-hoc search queries on any field can be executed either through CQL or through the Solr HTTP API. It is recommended to deploy search nodes separately from the Cassandra data nodes to achieve workload segregation and prevent competition of the resources. The Search Service nodes are strongly fault-tolerant, highly available, and linearly scalable as they use Cassandra under the hood.

MongoDB

MongoDB supports queries that perform a text search of the string content using a text index. Prior to performing text queries, you need to create an index for the target collection and specify the document fields that will be indexed. Any document field, which value is a string or an array of strings, can be indexed. Only a single text index per collection is allowed, but the index can cover multiple fields or all the collection text content.

MongoDB does not sort the results by default, but the text search queries compute a relevance score for each resulting document, and you can explicitly project the text score field and ask the system to sort it. A text search is available in the aggregation pipeline. MongoDB supports a text search for various languages. In addition, MongoDB Enterprise integrates the Basis Technology Rosette Linguistics Platform (RLP), allowing to perform normalization, word breaking, sentence breaking, stemming, or tokenization for a set of additional languages.

Summary

Both MongoDB and Couchbase Server deliver the full-text search capabilities out of the box. However, Couchbase Server implements a dedicated Search Service that leverages all of the advanced maintenance capabilities (scaling, replication, and load balancing) and supports a wider set of enterprise features. Cassandra does not have a built-in full-text search, but DataStax provides integration with Apache Lucene and Apache Solr—one of the most powerful combination for text indexing and search.

Table 3.16. The full-text search capabilities of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
7	9	9

3.3.4 Mobile devices support

Mobile devices support refers to an application support on both the mobile client and database server sides.

Couchbase Server

Couchbase Mobile is a dedicated solution for iOS, Android, and Windows-based mobile applications, as well as any Linux or Windows embedded system. The solution comprises an embedded NoSQL database (Couchbase Lite), a synchronization middleware service (Sync Gateway), and a distributed database (Couchbase Server).

Couchbase Lite manages and stores data locally on a mobile device. It can work either as a standalone instance or in a peer-to-peer (P2P) network. The P2P network allows a mobile device to accept connections from other devices running Couchbase Lite and exchange data with them. A Couchbase Lite user can modify data offline, and this data will be synced with the Couchbase Server or a P2P network via Sync Gateway once online.

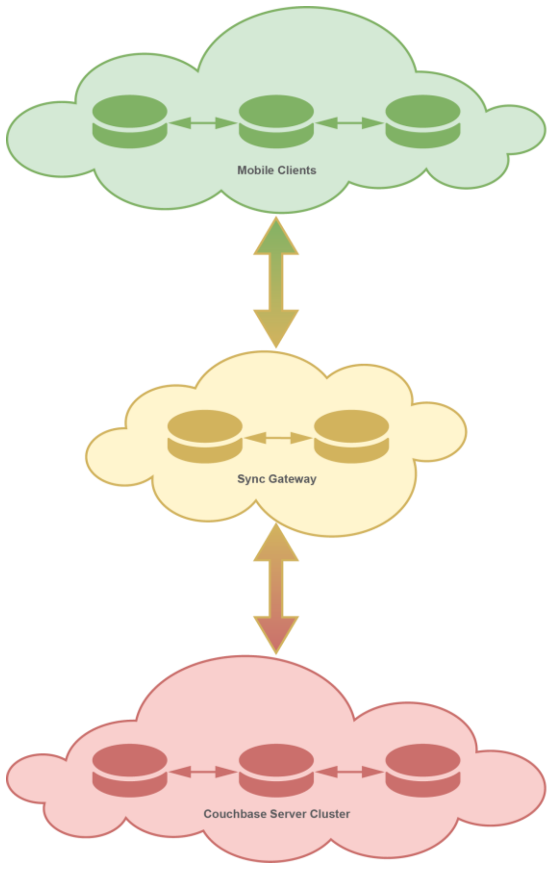


Image 3.18. The Couchbase Mobile stack

Cassandra

Cassandra does not provide any enterprise-level products for mobile devices as its data model does not fit the nature of interactive embedded applications.

MongoDB

MeteorJS, an open-source full-stack JavaScript platform for web, mobile, and desktop application development, made an attempt to provide a fully compatible MongoDB interface in a client application. The Minimongo library uses a publish-subscribe mechanism to synchronize its state with a MongoDB cluster. Unfortunately, the system works in a consistent manner only for small-scale, simple projects.

Summary

Unlike Cassandra and MongoDB, Couchbase provides a dedicated solution for mobile devices, which takes care of data consistency, security, synchronization, and availability regardless of the network connection. The MongoDB community offers several mobile-oriented libraries with a MongoDB-compatible interface, though with limited functionality.

Table 3.17. Mobile devices support of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
4	10	X

3.3.5 Logging and statistics

Logging and statistics refer to capabilities that log performance and errors, and conduct statistical analysis.

Couchbase Server

Couchbase Server creates a number of different textual log files for various system components. It logs information about cluster access, storage engine operations, indexing, replication, etc. Log files are automatically rotated and compressed. It is possible to change a log file location and log levels by modifying the Couchbase Server configuration files. Logs are accessible with either the CLI utilities—cbcollect_info, or the Couchbase Server REST API.

Couchbase Server aggregates a large set of numerical metrics on hardware resources usage (CPU, RAM, I/O), usage and performance numbers for buckets and vBuckets, views, as well as index and replication statistics (DCP, TAP, and XDCR). This information is available via administration interfaces: the CLI cbstats tool, the REST API, and the Couchbase Web Console.

Cassandra

With Cassandra, you can configure a format, a name, a size, a location, and a rotation of a logging file, as well as a logging level for a particular component or for the entire cluster. Logging levels can be updated in a runtime using the nodetool command-line utility or via the JConsole tool.

Cassandra provides plenty of statistics about network operations and connections, the number and status of database threads, keyspaces, and tables metrics, including read/write latencies, rows sizes, columns counts, number of SSTables, and other metrics. All these statistics are accessible through the nodetool, JConsole, and the DataStax OpsCenter Web UI.

MongoDB

A standard practice for MongoDB is log rotation. It includes archiving a current log file and starting a new one. MongoDB logs hold all necessary information for auditing and control. Each log message has one of the following severity levels: fatal, error, warning, information, and debug. It also contains the corresponding verbosity level specified on a per-component basis. These functional components include access control, database commands, control activities, the diagnostic data collection mechanism (FTDC), geospatial shapes parsing, indexes, network activities, queries, replication, sharding, storage, etc.

Performance metrics can be captured by enabling a database profiler that gathers comprehensive information about the executed reads and writes, cursor operations, and the issued database commands. The profiler stores this data in the system.profile capped collection.

Summary

All the three solutions aggregate a wide set of statistics and make them accessible. Each of the considered databases features various levels of logging and provides comprehensive information for performance and error analysis.

Table 3.18. Logging and statistics in NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	10

3.3.6 Integration

Integration refers to the ease of linking the common distributed computing solutions and software applications together into a single system.

Couchbase Server

Couchbase Server provides a number of connectors that exchange data with external platforms. For example, The Hadoop Connector streams keys into the Hadoop Distributed File System. The Kafka Connector streams data from Couchbase Server into Kafka, as well as publishes data from the Kafka topics into Couchbase Server. The Spark Connector provides integration with Apache Spark. The Elasticsearch Transport plugin provides the topology-aware real-time data replication to Elasticsearch. Applications can connect to Couchbase Server through the corresponding drivers available for popular programming languages.

Cassandra

Cassandra is integrated with several Apache Software Foundation projects, such as Hadoop, Spark, Kafka, and Solr. DataStax provides its own implementation of Spark integration for real-time and batch analytics capabilities and Solr integration to support full-text search queries. Interactions between client applications and Cassandra clusters are performed using drivers for various programming languages provided by DataStax and third-party vendors.

MongoDB

MongoDB has a connector for the Hadoop ecosystem, which allows you to use MongoDB as an input/output for batch data processing and analysis. The real-time processing capabilities are available through the Apache Spark integration. Data replication from MongoDB to Elasticsearch, Solr, and other systems is implemented in the *mongo-connector* project. The MongoDB connector for business intelligence (BI) enables users to visualize their MongoDB data using the existing relational BI tools. Numerous other open-source projects provide their own integrations with MongoDB.

Summary

All the three systems under consideration provide integration through plugins and connectors for the most common big data platforms implemented by NoSQL vendors, third parties, and open-source community members. All the solutions support popular programming languages through dedicated drivers.

Table 3.19. Integration capabilities of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	10

3.3.7 Documentation

Documentation refers to user guides, white papers, online help, quick-reference guides, etc.

Couchbase Server

All Couchbase Server versions are well-documented. The official documentation includes an under-the-hood description of the Couchbase Server architecture, complete guides for developers and administrators, as well as the API and query language references.

In addition, Couchbase maintains a blog with many of the company's staff members contributing relevant articles. The blog covers the architecture-related topics, upcoming release details, as well as developer-related concepts, such as best practices, code samples, and modeling approaches.

Cassandra

The documentation for the latest version of Cassandra is available on its official website. It covers major developing and operational topics required to employ the system. The documentation for earlier Cassandra versions can be found on the DataStax web portal. It contains exhaustive information about the Cassandra architecture, query language, cluster deployment planning, troubleshooting, selecting hardware, and other important topics. However, DataStax is no longer contributing to the Cassandra documentation as of 2017.

DataStax Academy provides free online courses and video tutorials about Cassandra internals, best practices of data modeling, diagnostic tools usage hints, etc.

MongoDB

The comprehensive product documentation is available on the official MongoDB website. It includes all the required guidelines both for developers and administrators. Publicly available information also includes few technical and a plenty of marketing white papers, webinars, and user groups.

MongoDB University is a massive open online course (MOOC) resembling a learning platform. It offers its users plenty of online courses on two major tracks—for developers and DBAs. Both public and private trainings taught by MongoDB engineers are provided on demand.

Summary

All of the considered NoSQL solutions are easy to learn due to the comprehensive documentation and widely available learning materials.

Table 3.20. Documentation of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
10	10	10

3.3.8 Usability and support

Usability refers to the ease of effective use and learnability of the solution, as well as the provided support offered to users.

Couchbase Server

Couchbase Server provides both administrators and developers with all the necessary instruments. The embedded Web Console tool centralizes deployment, configuration, maintenance, recovery, and monitoring tasks. In addition, Web Console has an interface for running and monitoring queries, building indexes, visualizing query plans, etc.

Couchbase Inc. provides learning services available worldwide. The services are based on real use cases and are focused on providing skills relevant for real applications.

Couchbase delivers both community and commercial support. The Couchbase User Community provides free assistance to its members. Three commercial subscription service-level agreements (SLAs)—Silver, Gold, and Platinum—are available. All tiers offer the same support channels—e-mail, web, and phone—and permit unlimited number of cases. However, these channels vary in hours of operation and initial response time. Couchbase Forums are maintained both by the community members and the Couchbase team engineers.

Cassandra

DataStax provides a proper set of tools for Cassandra operators and programmers. In addition to monitoring and maintenance capabilities, OpsCenter allows you to automatically set up the entire cluster, schedule backups and recover from them, run anti-entropy repairs, etc. As a part of OpsCenter, the Best Practice service provides a set of cluster configuration recommendations. DataStax DevCenter is a desktop IDE for accessing Cassandra using its query language and visualizing its tables with relations between them.

Comprehensive online courses on the Cassandra architecture, data modeling, and operations are provided by DataStax Academy for efficient learning.

The Commercial Cassandra support contracts and services are available from DataStax and other third-party companies. In addition, there are several forums and channels for the community support.

MongoDB

It is relatively easy to deploy and maintain MongoDB when you use a single replica set and relatively complicated when you need a sharded cluster. Both Enterprise Advanced and Professional editions include MongoDB Cloud/Ops Manager that significantly simplifies administrative tasks. MongoDB Community Edition does not include GUI management tools out of the box.

MongoDB, Inc. supports the learning initiatives through MongoDB University, a MOOC-like educational platform for developers and administrators. MongoDB is a part of the MEAN (MongoDB, Express.js, AngularJS, and Node.js) stack—a popular full-stack framework that makes it possible to develop both server-side and client-side applications with JavaScript.

The comprehensive MongoDB support programs are provided both by MongoDB, Inc. and various third-party companies. There is a reasonably large user community on popular platforms, such as StackOverflow, ServerFault, the MongoDB user group "Community Support Forum," and the IRC channel on a freenode, where you can get help and exchange your ideas with others.

Summary

Both Couchbase Server and Cassandra provide SQL-like query languages that extremely simplify the learning curve for new users. All the three products have a web-based UI for cluster administrators. MongoDB is more complicated in its use because of the relatively complicated architecture concepts and a flexible but non-standard query language.

All the three considered solutions have large and responsive communities providing free support. Commercial support is available by the well-established companies.

Table 3.19. Enterprise and community support of NoSQL data stores on a scale of 1–10

MongoDB	Couchbase Server	Cassandra
9	10	10

4. Conclusion: 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 the 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 4.1 Comparative scores of 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		8		10		10	
2. Topology		7		9		9	
3. Scalability		8		10		9	
4. Consistency		10		10		10	
5. Availability		6		8		10	
6. Replication		9		10		10	
7. Fault tolerance		7		9		10	
8. Recovery		9		10		10	
9. Disaster recovery		10		10		10	
10. Backup		9		9		9	
11. Configuration management		9		10		9	
12. Maintenance		8		10		9	
13. Monitoring		9		10		10	
14. Structure and format		10		10		10	
15. A query language		8		10		8	
16. A full-text search		7		9		9	
17. Mobile devices support		4		10		X	
18. Security		9		9		9	
19. Logging and statistics		10		10		10	
20. Integration		10		10		10	
21. Documentation		10		10		10	
22. Usability and support		9		10		10	
Total Scores:		**Basic**	**Weighted**	**Basic**	**Weighted**	**Basic**	**Weighted**
		186		**213**		**201**	

Our general advice is to use Couchbase Server (or MongoDB) for storing and processing semi-structured data that can benefit from using the JSON format. You can also consider Couchbase Server for mobile/offline-first web applications. Cassandra works well for write-intensive workloads on big data-scale for structured data (including non-relational data).

Couchbase Server focuses more heavily on the vertical scalability than either Cassandra or MongoDB. Its multidimensional scaling feature targets specific customer scenarios that need to quickly scale up different workload types (either key-value, indexing, or full-text search). The concept seems attractive for deployments at a scale up to hundreds of cluster nodes. N1QL extends SQL providing extra functions for querying JSON documents. XDCR allows simple and reliable design for multi-cluster deployments with advanced topologies.

Cassandra is the only AP storage system under consideration in this study. It provides a high level of availability and partition tolerance. Thus, Cassandra is capable of running on thousands of node clusters and is well suitable for accepting high throughput write workload of non-frequently changing, log-oriented data. It provides a good number of tunable consistency levels, including those specifically designed for multi-datacenter deployments.

MongoDB scores lower primarily due to its hierarchical architecture. It features a moderate number of cluster components—each having its own configuration features, explicit master-slave replication decoupled from data partitioning, and exceptional role of the configuration servers. All the mentioned points significantly complicate the database deployment design and maintenance and impose additional requirements on the client code to avoid durability problems. However, if properly used, MongoDB enables a strong level of consistency and flexible partitioning strategies. The winning point for MongoDB is its popularity and availability of learning materials.

5. About the Authors

Vladimir Starostenkov is a Senior R&D Engineer at Altoros. He is focused on implementing complex software architectures, including data-intensive systems and Hadoop-driven apps. Having background in computer science, Vladimir is passionate about artificial intelligence and machine learning algorithms. His NoSQL and Hadoop studies were published in NetworkWorld, CIO.com, and other industry media.

Nikita Gorbachevski is a Senior Java Engineer at Altoros, specializing in distributed systems. He maintains a highly proficient knowledge profile in concurrent and parallel computing, highload database systems, and the blockchain technology.

Mikalai Lushchytski is a committed Java developer with comprehensive knowledge in designing, implementing, customizing, upgrading, and maintaining highload enterprise applications.

***Altoros** brings Cloud Foundry-based "software factories" and NoSQL-driven "data lakes" into organizations through training, deployment, and integration. With 300+ employees across 9 countries in Europe and Americas, Altoros is the company behind some of the world's largest Cloud Foundry and NoSQL deployments. For more, please visit www.altoros.com.*

To download more NoSQL guides and tutorials:

- *check out our resources page,*
- *subscribe to the blog,*
- *or follow @altoros for daily updates.*