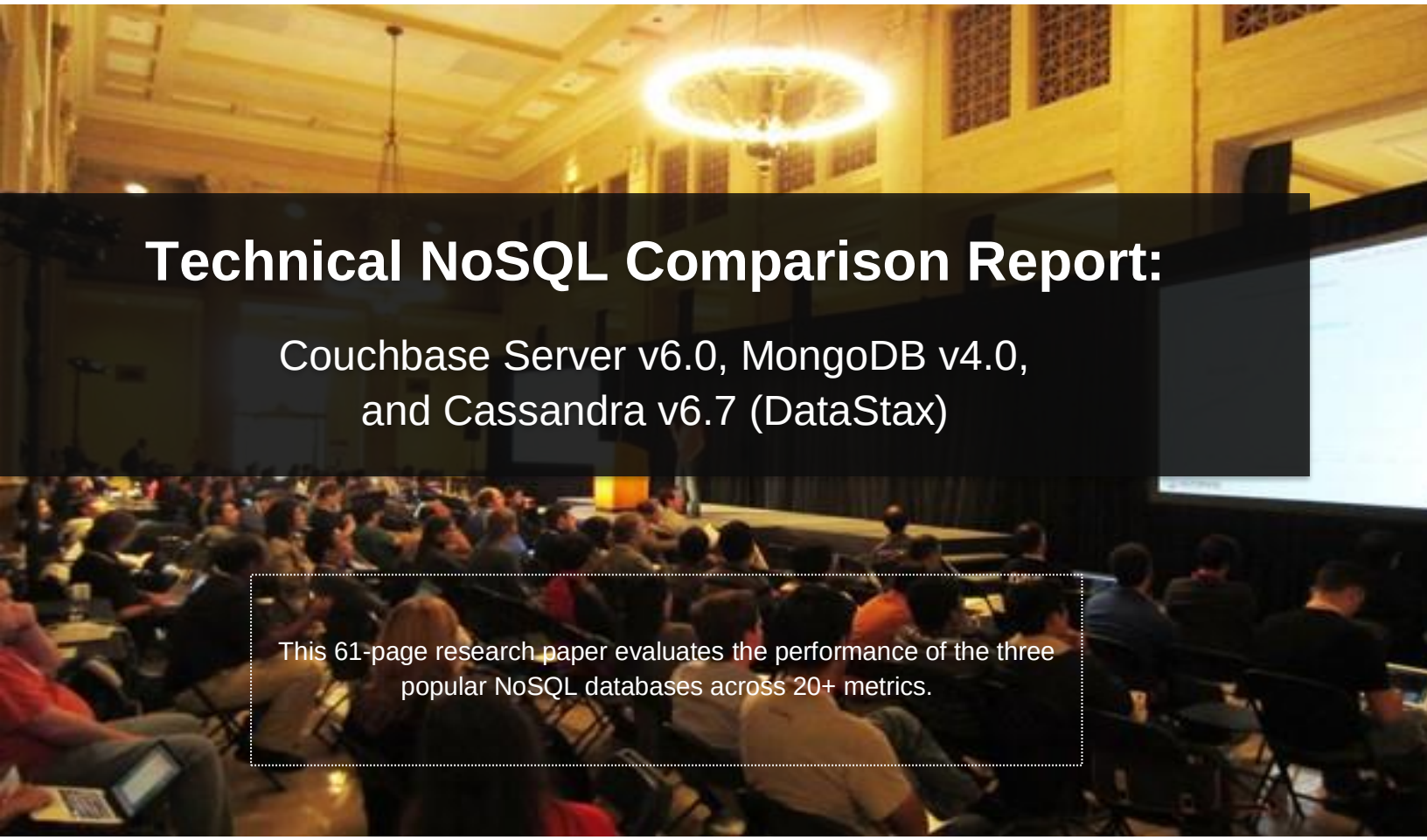
ALTOROS™



Technical NoSQL Comparison Report:

Couchbase Server v6.0, MongoDB v4.0, and Cassandra v6.7 (DataStax)

This 61-page research paper evaluates the performance of the three popular NoSQL databases across 20+ metrics.

Couchbase mongoDB Cassandra

By **Artsiom Yudovin**, Data Engineer at Altoros,
and **Yauheniya Novikova**, Sr. Java Engineer at Altoros.

Q2 2019

Table of Contents

1. Executive Summary

A variety of NoSQL solutions exist to solve the problems of rapidly growing data sets, data structure organization, as well as management and efficiency of accessing data. The implementation approaches of databases vary significantly from a vendor to a vendor, so it is important to recognize the strengths and weaknesses of the various NoSQL options available. This is why, before deciding which NoSQL database to use for a solution, system architects and IT managers typically compare data stores in their own environments, employing representative data and user interactions for the expected production workloads.

This report provides an in-depth analysis of the leading NoSQL solutions: **Couchbase Server v6.0**, **MongoDB v4.0**, and **DataStax Enterprise (Cassandra) v6.7**. The comparison evaluates the systems from different angles to help choose the most appropriate 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 some recommendations on the best ways to configure, install, and use the NoSQL databases depending on their specific features. In the last chapter, a comparative table summarizes how Couchbase Server, MongoDB, and DataStax Enterprise (Cassandra) 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 procedures), scores were applied based on in-depth review of documentation and our own experience in development and production.

2. Installation and Configuration

This section evaluates how easy it is to install and configure the databases.

Couchbase Server

Couchbase documentation contains all the details of the installation process for Linux, Windows, and Mac OS. Couchbase Server provides a preconfigured Amazon Machine Image (AMI) in the Amazon Web Services marketplace for deploying on AWS. Couchbase Server Enterprise Edition Google Cloud Launcher allows for deploying on Google Cloud Platform (GCP); the Azure Resource Manager (ARM) template enables the deployment of Couchbase Server on Microsoft Azure.

Couchbase Server Docker images are available at the Docker Hub. In addition, Couchbase can be run and managed on Kubernetes and OpenShift by using Autonomous Operator. Couchbase Server offers three options for configuration: a *REST API*, a *command-line interface (CLI)*, and a *web UI*.

MongoDB

MongoDB provides various options for cloud and on-premises installations. The database can be deployed using Atlas, a cloud solution for AWS, Azure, and GCP. Besides the cloud offering,

distributions for specific operating systems, as well as Ops Manager and Cloud Managers are available.

To use Ops Manager, one should meet the deployment prerequisites, which imply configuring networking access for hosts that serve MongoDB deployments, as well as installing MongoDB Enterprise dependencies and Automation Agents on each machine in the cluster. Before installing Ops Manager, a database for Ops Manager and an optional backup database should be created. Once completed, a MongoDB cluster can be deployed via the web UI or Kubernetes.

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

Sharded cluster infrastructure requirements and complexity call for careful planning, execution, and maintenance. Before deploying a production cluster, database engineers should think carefully about the cluster architecture and make decisions regarding cluster topology, security, and backup. In addition, one has to answer the questions related to the amount of primaries (data shards), secondaries, arbitries, as well as the number of mongooses and where they will be launched. Furthermore, developers have to think about the configuration of cluster members and storage engine type.

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.

DataStax Enterprise (Cassandra)

DataStax Enterprise (DSE) requires preinstallation of the Java Development Kit. There are two ways to install DataStax Enterprise (Cassandra):

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

DataStax Enterprise (Cassandra) should be installed and configured on each node in a cluster. The database provides the following options for configuring a cluster:

- property files (cassandra.yaml)
- DataStax OpsCenter web UI

DataStax Enterprise (Cassandra) supports deployments on Amazon Web Services and Microsoft Azure. In addition, DataStax Enterprise (Cassandra) images are available at the Docker Hub.

Summary

DataStax Enterprise (Cassandra) and Couchbase Server provide an administration console for configuring a cluster in a single place, making it easier to install them. The configuration of a MongoDB cluster is a bit more complicated than the one of DataStax Enterprise (Cassandra) and Couchbase Server.

Table 2.4 Ease and convenience of installation on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	8	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 either fixed or flexible. Different cluster topologies lead to different internal and external data flows and different levels of system scalability and availability.

Couchbase Server

Couchbase Server handles a variety of different workloads ranging from document storage and caching, through various types of indexing and querying to server-side scripting. A cluster is architected in such a way that these workloads are broken into the following logical "services" that are then distributed across the nodes.

- The Data Service is the core document storage component, provides strongly consistent access to documents via memory and/or disk. The service must run on at least one node of any cluster.
- The Query Service is an engine to parse, process, and handle the N1QL language queries (SQL-like). The service leverages the Index Service (and in the future, the Search Service) to optimize queries for low-latency and high-throughput execution.
- The Index Service creates and maintains primary and secondary indexes on a data set within Couchbase Server. Indexes are maintained asynchronously and can be made strongly consistent per application request.
- The Search Service provides full-text search (i.e., fuzzy matching, stemming, tokenization, etc.) through the use of inverted indexes.
- The Analytics Service is a parallel data management (MPP) engine for running complex analytics queries. The service is accessed via the same N1QL language as the Query Service and is designed for unindexed (i.e., ad-hoc) queries and/or processing much larger result sets than the Query Service.
- The Eventing Service is a V8 JavaScript engine for near real-time, asynchronous, server side handling of data change "events." Similar to post-triggers in traditional databases.
- The Cluster Manager is responsible for all cluster-wide operations, such as monitoring a cluster state and generating statistics, performing rebalancing operations to evenly distribute data over the nodes, keeping clients synchronized with the topology, handling authentication and logging, etc.

Each node always includes the Cluster Manager, but other services can be enabled and configured at run time depending on the application access, performance, and reliability requirements. This allows the above workloads to be isolated in terms of both resource usage and a method of scaling. For instance, the Data Service tends to be better suited to more, smaller nodes, whereas the Index Service is better suited to fewer, larger capacity nodes. It is also often preferable to scale one service separately from another, because it enables an engineer to increase the Query Service capacity without shuffling or moving data stored within the Data Service.

The service layout and topology, as well as hardware allocation, can be changed on the fly without disrupting any application behavior. Each service handles its own distribution, replication, and high availability according to a preferable configuration. (The Data Service is transparently sharded and replicated, while the Query Service is load-balanced and stateless, etc.)

All of these services communicate with each other through the Database Change Protocol (DCP), which was designed to quickly and efficiently move large amounts of data. DCP also feeds the cross-datacenter replication (XDCR) functionality, which allows for setting up data replication between two or more remote clusters. XDCR supports both unidirectional (active-standby) and bidirectional (active-active) replication for centralized, hierarchical, ring, or any user-defined multi-cluster topologies.

The client libraries of Couchbase Server connect directly to cluster nodes that are responsible for servicing each request. (Reads and writes go directly to the Data Service, N1QL queries directly to the Query service, etc.) The clients handle all of the connection pooling and topology awareness (i.e., the number of nodes and placement of services), so that there is no difference in behavior when connecting to or moving between different topologies. The same applies to any administrative access or maintenance, a single node cluster with all services enabled (e.g., on a laptop) will exhibit exactly the same behavior as a 100-node cluster with each service isolated from one another.

Couchbase Server supports two basic topologies:

1) **Co-located services**: All the nodes have equal responsibility. A workload is distributed equally across a cluster.

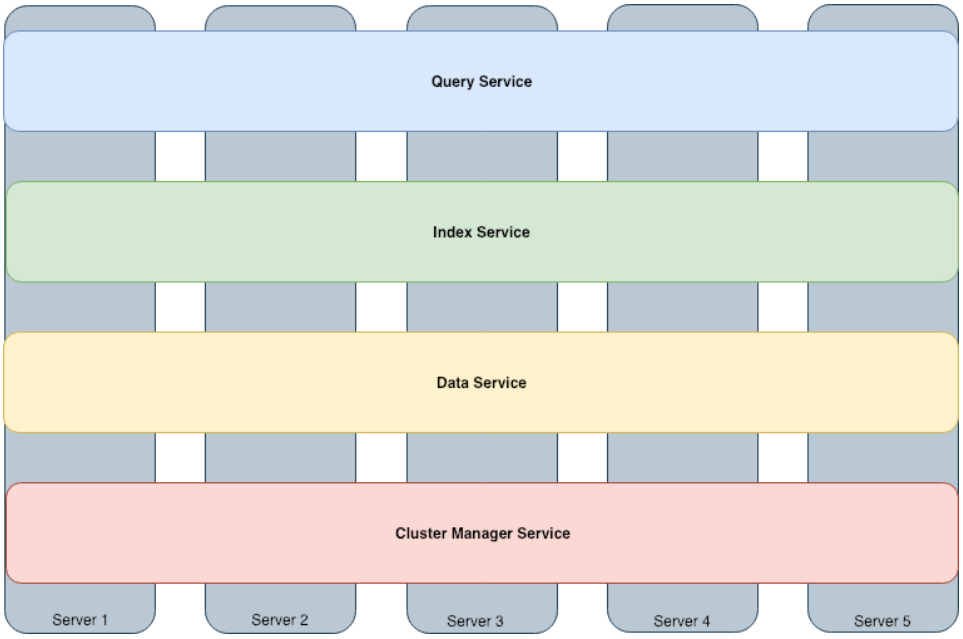


Figure 3.1.1.1 The Couchbase Server cluster topology

This model is well suited to smaller clusters with lower workload and data volume requirements. As a workload or data volumes increase, this mode has a few drawbacks:

- All the processes of core data operations (inserts, updates, and deletes), as well as index maintenance and executing queries compete for the same resources.
- Services cannot be optimized individually, because all the nodes will be used equally.

2) **Multi-dimensional scaling**: Different nodes have different responsibility.

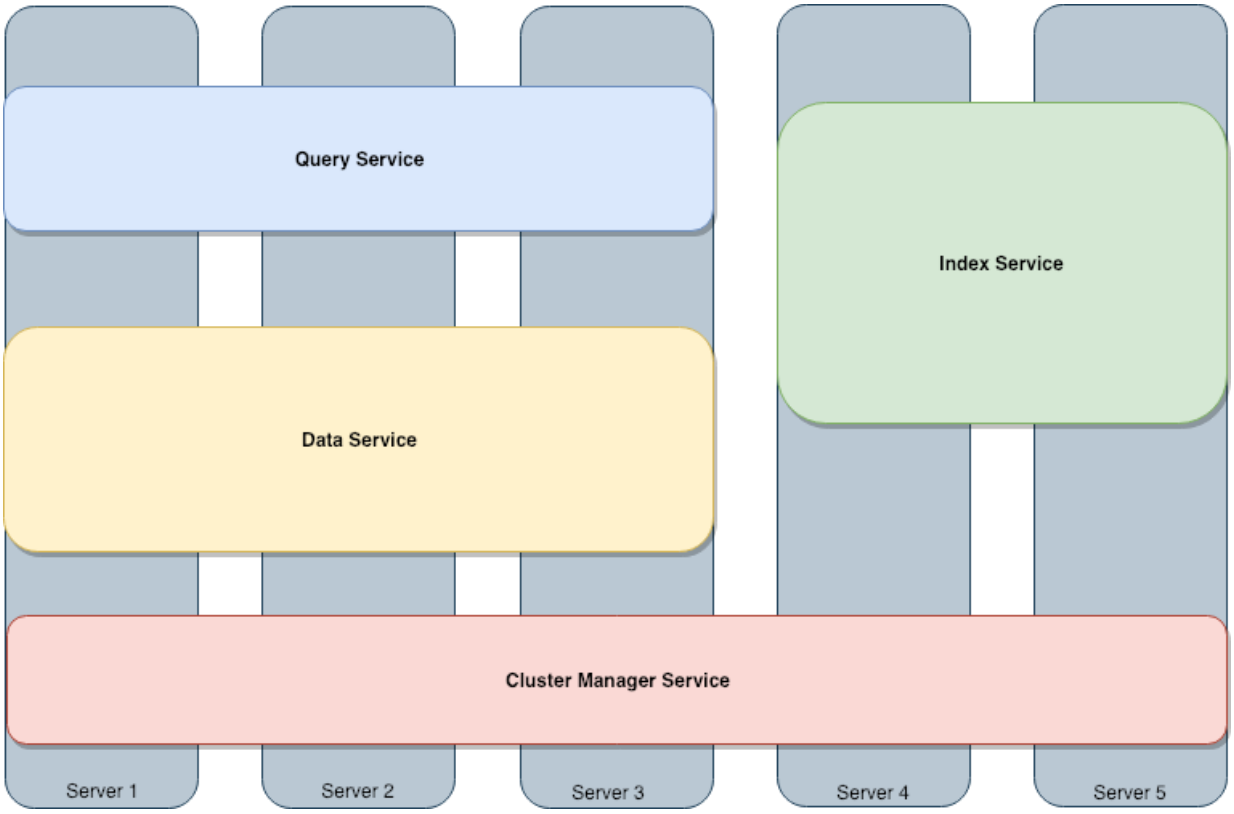


Figure 3.1.1.2 A sample Couchbase Server cluster topology

This model is recommended for production, as it allows for workload isolation and better resource optimization. Even in this mode, all nodes have the Cluster Manager service, which ensures a common management plane.

MongoDB

MongoDB employs a hierarchical cluster topology that combines router processes (*mongos*), config servers (metadata), and data shards. A number of data shards together with configuration servers and mongooses are combined into a sharded cluster.

Shards contain a subset of sharded data. Configuration servers store metadata that reflects a state and organization for all the data and components within a cluster, as well as such authentication configuration information as Role-Based Access Control or internal authentication settings for a cluster. MongoDB uses configuration servers to manage distributed locks.

Clients do not directly communicate with data shards—a *mongos* is responsible for providing an interface between a client app and a sharded cluster, serving as a query router. The *mongos* instances cache and use metadata from configuration servers to route read and write operations to the appropriate shards. A *mongos* has no persistent state and consumes minimal system resources.

The most common practice is to run *mongos* instances on the same systems as application servers, but it is possible to maintain *mongos* instances on the shards or on some other dedicated resources.

Configuration servers and shard servers must be running as a replica set. A replica set can have a single primary, a single or more secondaries, and an optional arbiter node. A replica set can have up to 50 members, but only seven voting members.

Arbiters are *mongod* instances that are part of a replica set, but do not hold data and cannot become a primary. An arbiter node is added in case a replica set has an even number of members. When the current primary goes down, arbiters participate in elections to add a vote for a primary node in order to break ties if a replica set has an even number of members. Arbiters have minimal resource requirements and do not need dedicated hardware. An arbiter can be deployed on an application server or a monitoring host.

The primary nodes receive all write operations on the primary and then record the operations on the primary's *oplog*. Secondaries copy and apply operations from the primary's oplog in an asynchronous process to maintain an identical data set to its own. By default, an application directs its read operations to the primary member, and clients cannot write data to secondaries but can direct read data from secondary members, if an appropriate *read preference* is set up for secondary members. A secondary can become a primary. If the current primary becomes unavailable, the replica set holds an election to choose which of the secondaries to become the new primary.

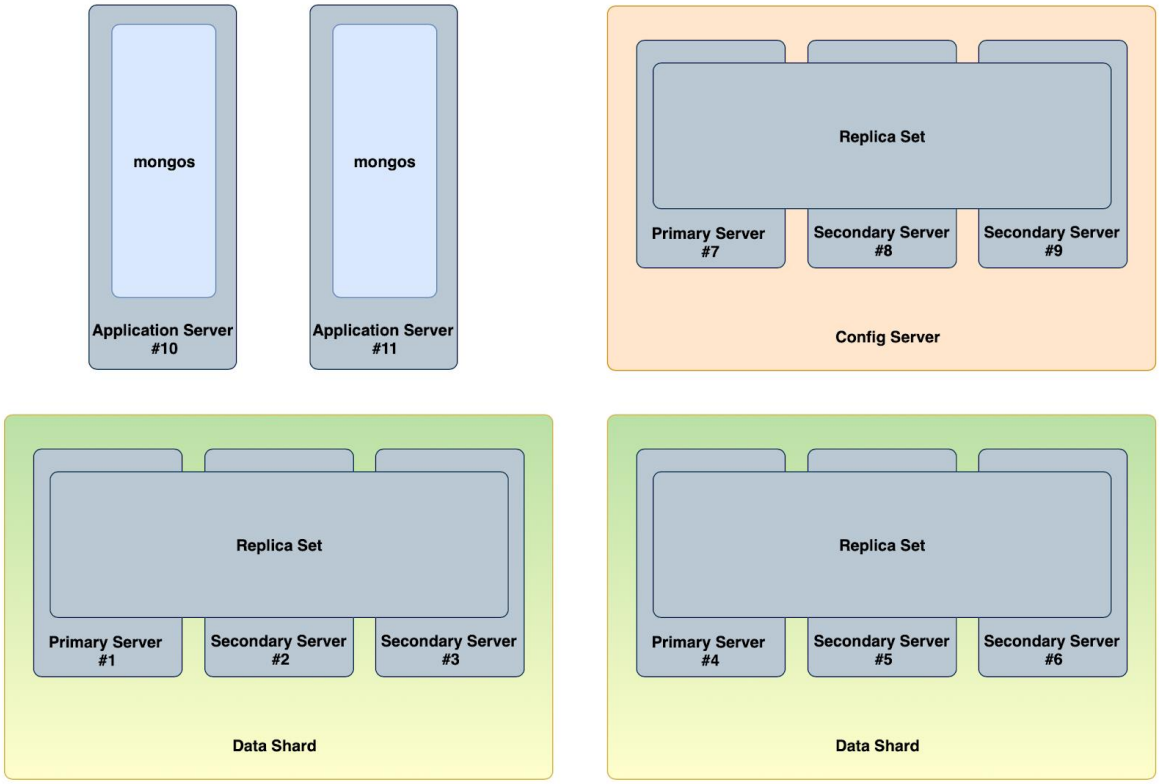


Figure 3.1.1.3 A sample MongoDB cluster topology

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. These 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 the *active-standby* pattern or get distributed evenly across multiple data centers for the write workload equalization.

Tag aware sharding enables users to control data distribution in a multi-datacenter MongoDB cluster by tagging ranges or hashes of a shard key to a single or more data shards. The database automatically computes the hashes when resolving queries using hashed indexes, and applications do not need to compute hashes. The resulting specific deployment architecture (together with the reinforced application-side logic) allows to implement distributed local writes.

DataStax Enterprise (Cassandra)

DataStax Enterprise (Cassandra) is a distributed cluster system with a peer-to-peer topology. Data is evenly distributed across all the nodes. Cluster nodes communicate with each other via the *gossip protocol*. The protocol is used to exchange information about the state of nodes. Some nodes must be reached out to through a seed node, which is used to bootstrap the gossip process for new nodes joining a cluster. Data center and rack awareness are provided by the snitch process, which enables efficient dynamic request routing and smart replica distribution.

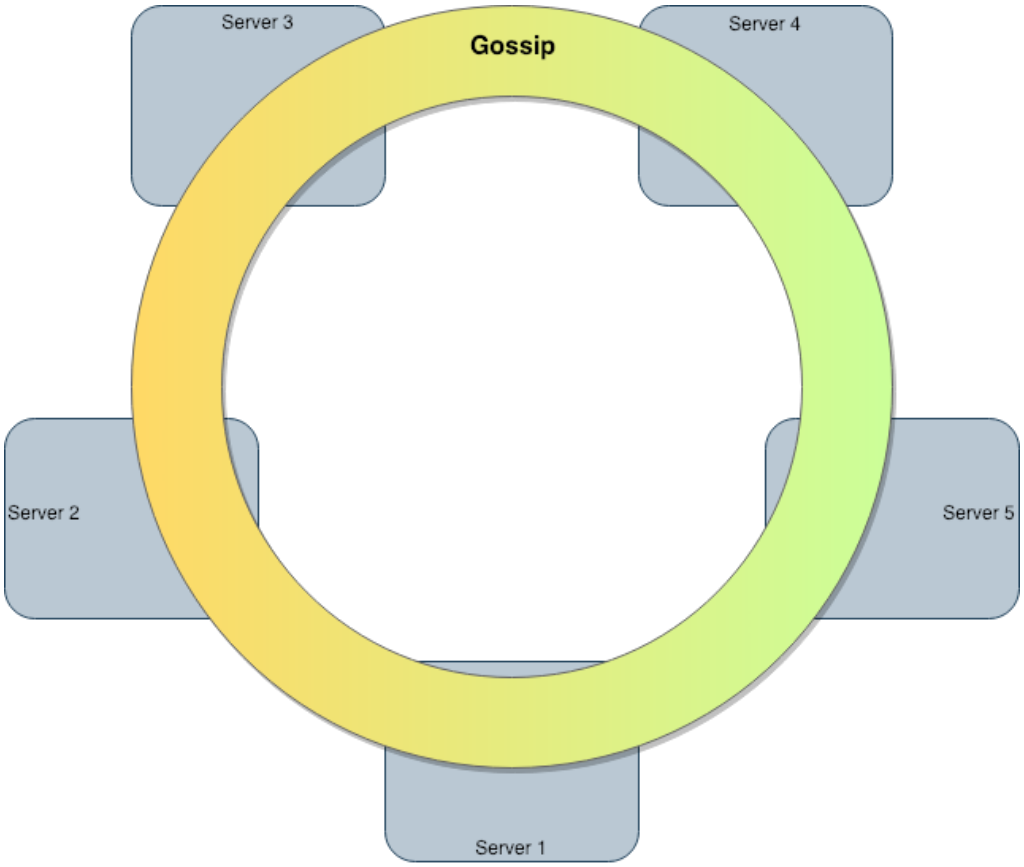


Figure 3.1.1.4 A sample DataStax Enterprise (Cassandra) cluster topology

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

Summary

DataStax Enterprise (Cassandra) makes use of a peer-to-peer architecture. The result is that a Cassandra cluster does not have a single point of failure. This type of an architecture provides a good quality of scalability and simplicity of use. The Couchbase Server cluster structure has similar design origins—no single point of failure. At the same time, its Multi-Dimensional Scaling architecture enables granular control over workload isolation as an alternative to the pure equal workload distribution. MongoDB has a master-slave architecture more similar to legacy databases—therefore, the resulting cluster has a single point of failure.

All databases support multi-datacenter and multi-cluster topologies, however MongoDB cannot truly support an active-active data center setup. It is worth mentioning that providing no significant benefits compared to other databases, the MongoDB architecture is more complex, less encapsulated, and requires more efforts for deployment and configuration.

Table 3.1.1 Topology of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	7	8

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 in which 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 a 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 a cluster, whereas in heterogeneous topologies, you add resources only to the nodes that support specific workloads you want to increase.

Couchbase Server

Couchbase Server offers two scaling models: a *homogenous scaling* model and an *independent scaling* model. Each of these models can support horizontal and vertical scaling.

In a *homogenous scaling* model, a workload is distributed equally across a cluster. Each node is responsible for a similar slice of the workload. Services will compete with each other, and it will be impossible to optimize the resource utilization of each service to your needs.

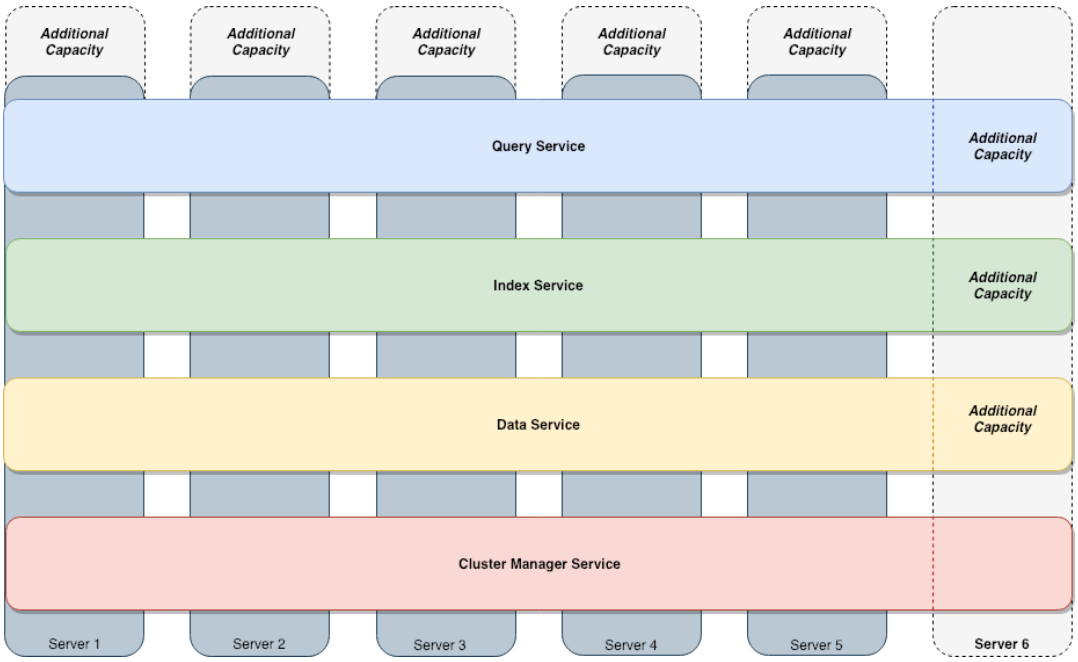


Figure 3.1.2.1 The homogeneous scaling model of Couchbase Server

Multi-Dimensional Scaling provides flexibility for improving performance by enabling independent expansion of these services and the corresponding workloads both vertically and horizontally. Such an option helps to minimize competition and interference between services. For example, if a workload requires a large volume of queries or particularly 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 a workload for all the services with a potential contention for computational resources between them.

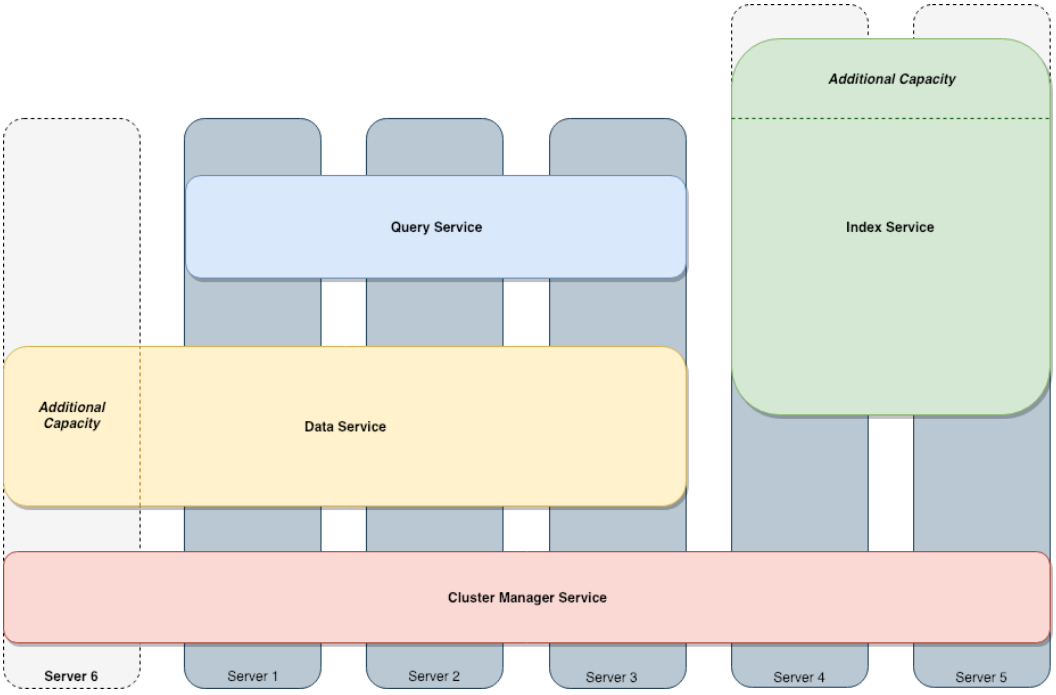


Figure 3.1.2.2 The independent scaling model of Couchbase Server

Each of the services scales both vertically and horizontally, but there are some recommendations for the optimal resource usage:

- 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 and Eventing services are CPU-intensive and have very low RAM or disk requirements. From the performance perspective, there is not much difference between a lot of small nodes or fewer large nodes. These services are 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.
- The Analytics Service is responsible for running complex queries efficiently over many records. Such data processing requires high CPU usage and disk throughput, consuming large amounts of memory, if available. Due to the typical task of isolating analytical processing from operational workloads, the Analytics Service should be deployed independently, with no other Couchbase services on the same node.

MongoDB

MongoDB has two complementary strategies for scalability:

1) Scaling 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 the replica set, you can achieve almost linear scalability for read operations but remain bottlenecked to a single node for writes.

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 a cluster configuration increases system complexity and can reduce the application's flexibility. Therefore, data shards are typically kept equal in size.

Scaling replica sets and increasing the number of shards might not directly improve the database throughput, so additional *mongooses* may be required.

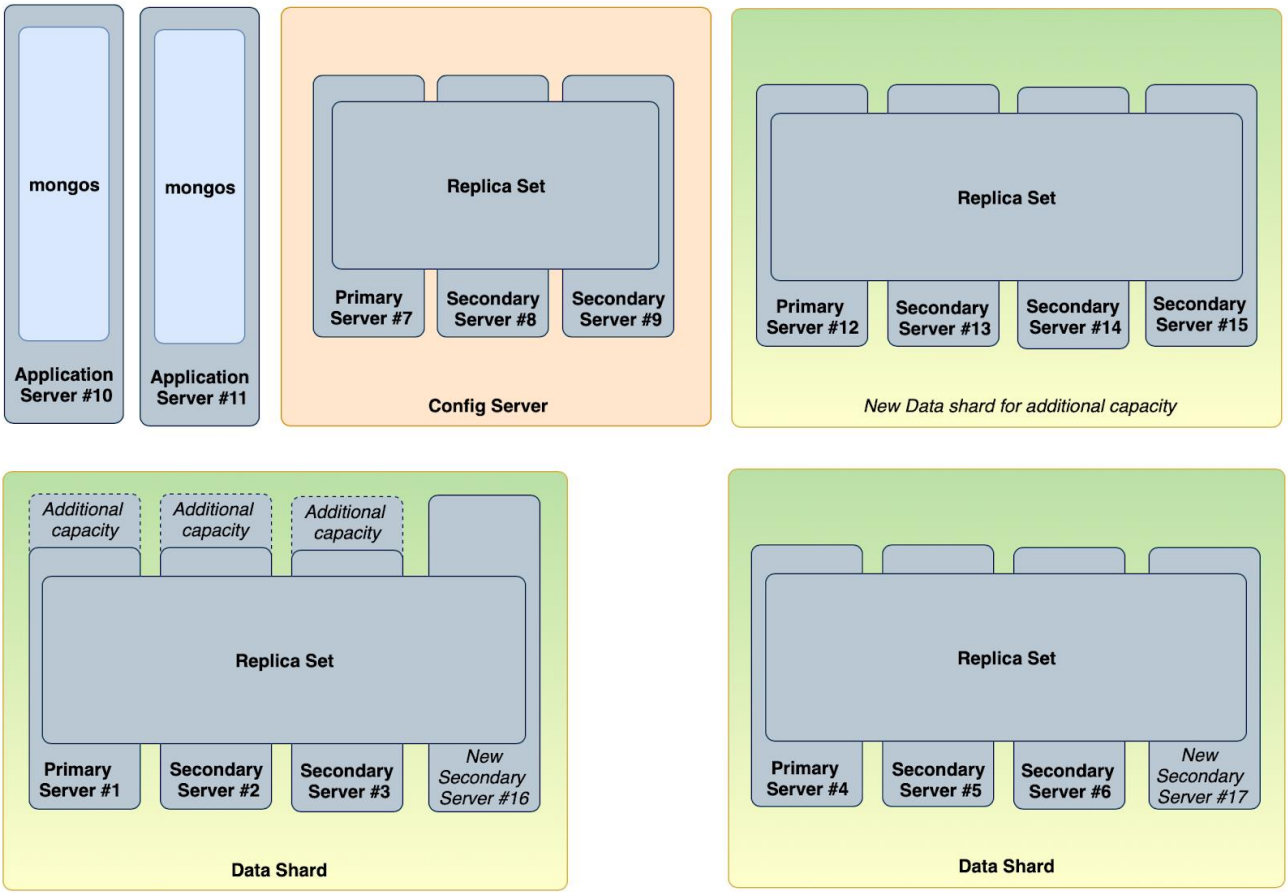


Figure 3.1.2.3 Scaling in MongoDB

DataStax Enterprise (Cassandra)

Cassandra features almost linear horizontal scalability. The 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.

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. The rest of the allocated memory is taken by the OS file system and off-heap caches. DataStax Enterprise (Cassandra) performs intensive parallel computing, so multi-core CPUs are also recommended. Local hard drives are preferable over network attached storage.

Traditionally, DSE Analytics is deployed on the same machine as DSE (Cassandra). Such a deploying process results in DSE Analytics and the database itself competing and interfering with each other. In addition, Cassandra provides an option for organizing independent scaling model of DSE Analytics. This option is called DSE Analytics Solo, and it offers to deploy DSE Analytics processing on other hardware configurations in a different data center. In this case, DSE Analytics and Cassandra will scale separately, preventing competition and interference with each other.

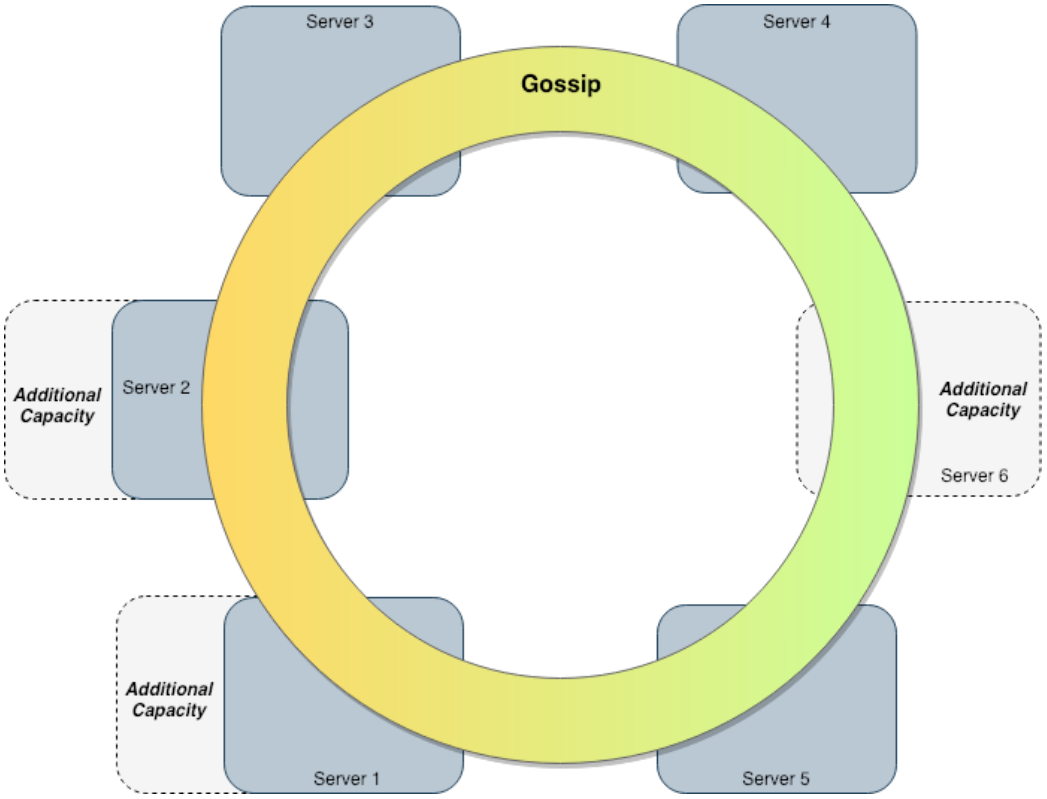


Figure 3.1.2.4 Scaling in DataStax Enterprise (Cassandra)

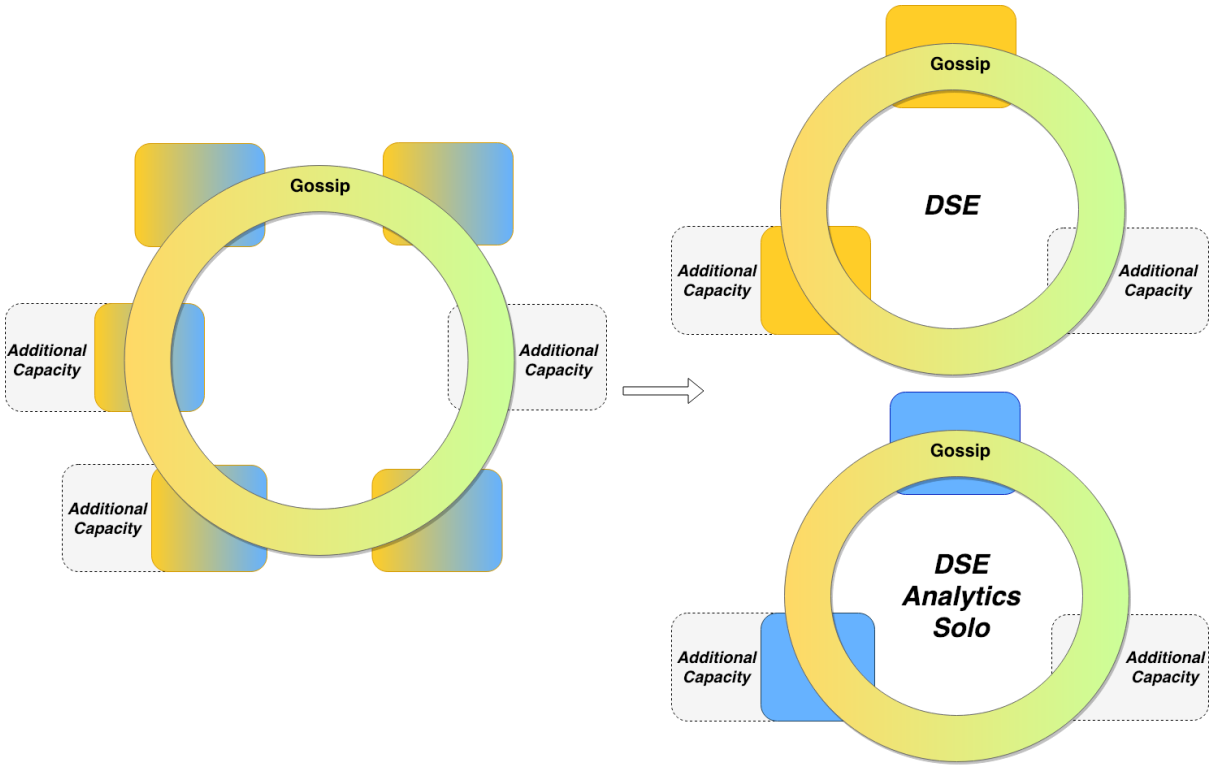


Figure 3.1.2.5 DSE Solo in DataStax Enterprise (Cassandra)

Summary

Couchbase Server and DataStax Enterprise (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 a cluster design to achieve the same database efficiency of scaling that Couchbase Server and DataStax Enterprise (Cassandra) ensure.

Table 3.1.2 Scalability of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	7	10

3.1.3 Replication

Replication is the process of copying and maintaining data in multiple physical and/or logical locations. This enables high availability and can improve database access performance.

Couchbase Server

Couchbase Server supports data replication between nodes of a cluster and between clusters in different data centers.

Intra-cluster replication is data replication across the nodes of a cluster. Couchbase data is divided into buckets, and vBuckets are created for each bucket. vBuckets enable distribution evenly across the memory and storage facilities of a cluster. Couchbase Server uses *Master Services* for the evenly distributing location vBuckets and minimizing data loss. There are two types of vBuckets:

- An active vBucket is responsible for read and write operations.
- A replica vBucket is responsible for replication data and is able to support read operations.

Replica vBuckets receive a stream of changes from an active vBucket using the *Database Change Protocol* (DCP). DCP maintains replica vBuckets, incremental MapReduce, spatial views, Global Secondary Indexes, XDCR, backups, and external connections. Data is asynchronously pushed from active to replica vBuckets. In a Couchbase Server client, it is possible to configure 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.

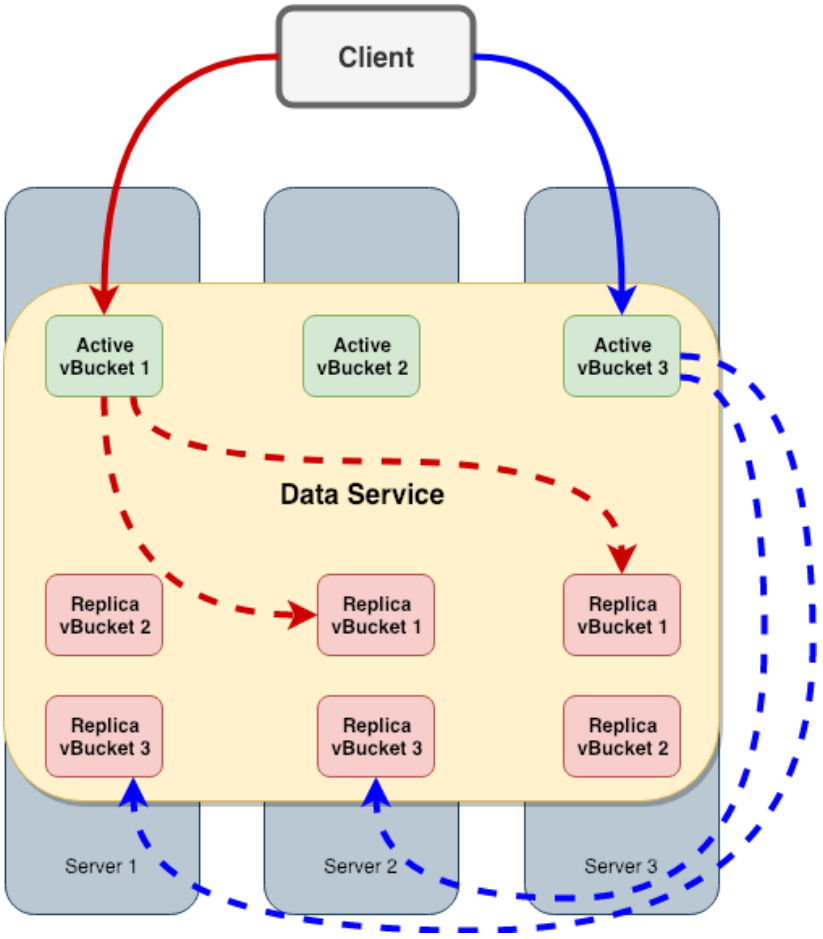


Figure 3.1.3.1 Intra-cluster replication in Couchbase Server

Cross-datacenter replication (XDCR) enables replication between clusters located in different locations. XDCR offers highly flexible replication:

- *Unidirectional* or *bidirectional* replications are supported by XDCR.
- Multi-master architecture supports *any topology*—such as start, ring, chain, mesh, one-to-many, many-to-one, etc.
- Depending on app requirements, buckets can be selectively replicated (*bucket-level* replication).
- Key-based *filtering* allows a further subset of data to be replicated. Advanced filtering (non-key based) is to be released and will strengthen this capability.

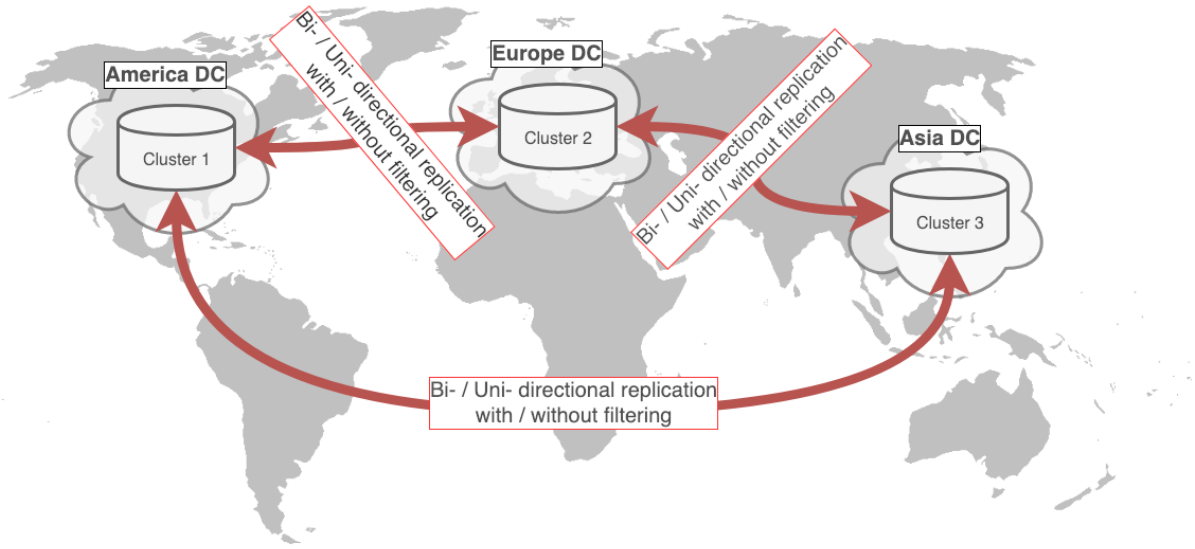


Figure 3.1.3.2 Multi-cluster replication by Couchbase Server

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.

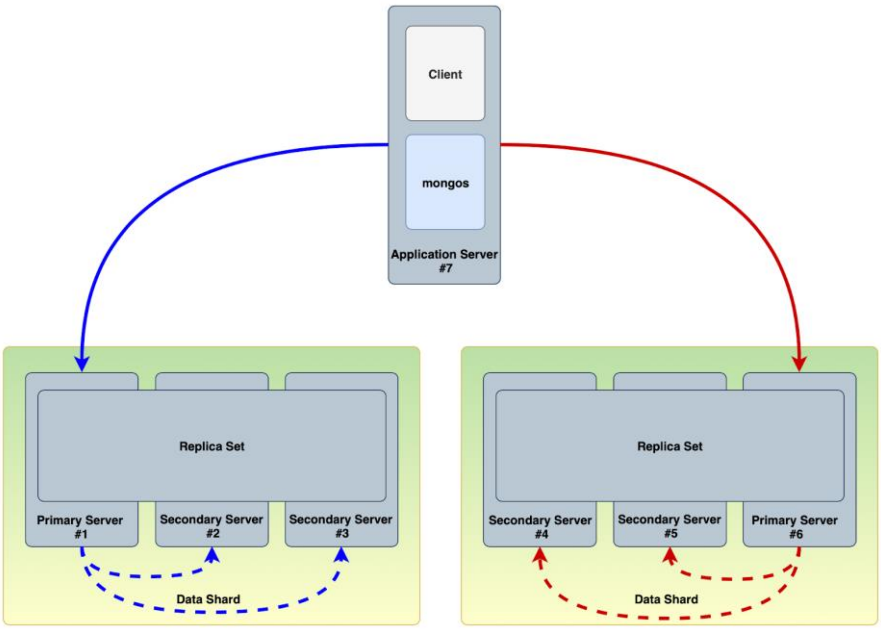


Figure 3.1.3.3 Intra-cluster replication in MongoDB

A replica set has an exclusive node, called *primary*, which 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. There are two types of multi-datacenter replication. The first one is *read locally and write globally*, which means each of the available data centers will be responsible for writing its portion of data (see Figure 3.1.3.4).

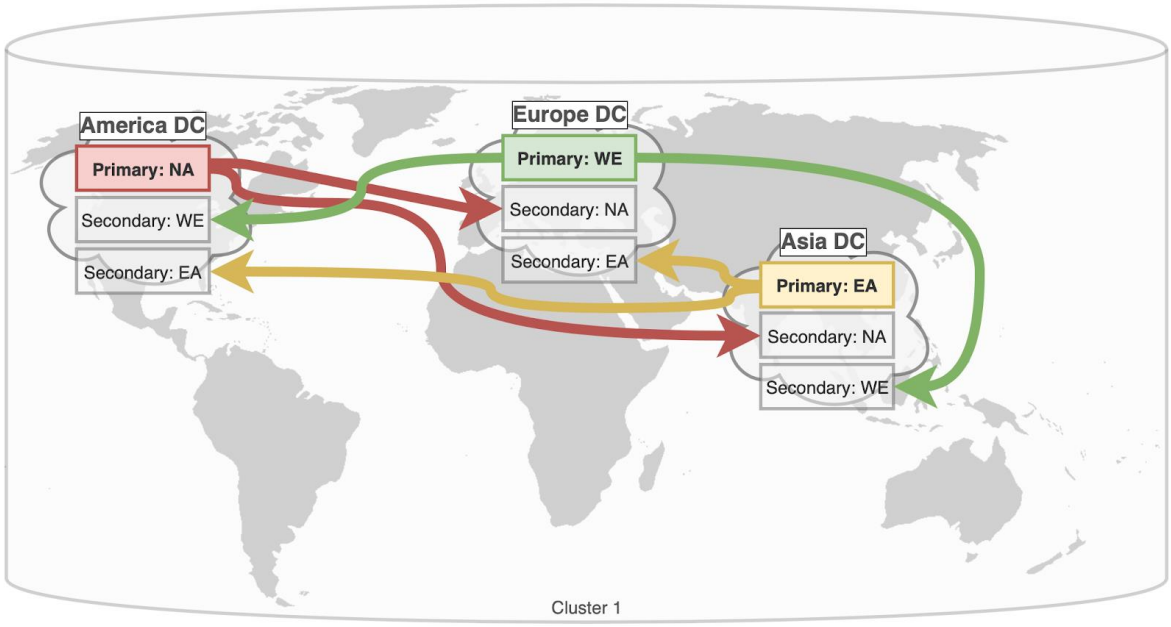


Figure 3.1.3.4 Multi-datacenter replication in MongoDB: read locally, write globally

The second type is *active-standby*, which means all data is written exclusively to a single identified data center (see Figure 3.1.3.5).

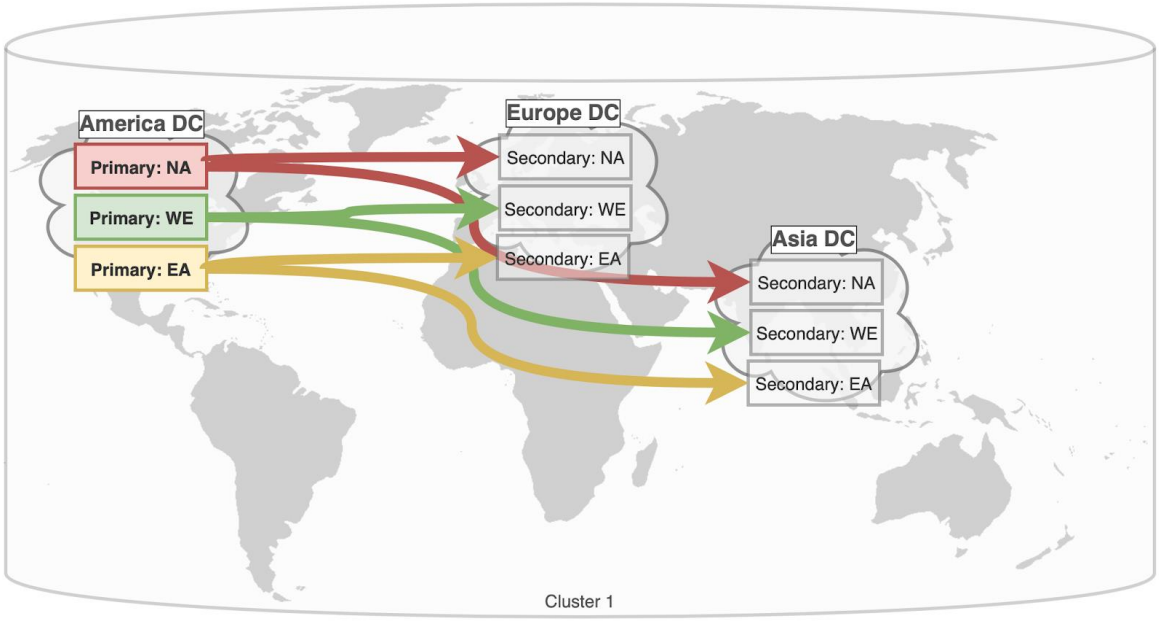


Figure 3.1.3.5 Multi-datacenter replication in MongoDB: active-standby

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 a target data set. The oplog itself is a circular buffer of a configurable size implemented as a special MongoDB capped (fixed-size) collection.

DataStax Enterprise (Cassandra)

DSE stores replicas on multiple nodes to provide reliability, high availability, and partition tolerance. A **replication strategy** determines the locality of replication in cluster. Cassandra supports two strategies. There is one for *single-datacenter* deployments and the other one for *multi-datacenter* deployments.

In addition, Snitches in Cassandra allow for spreading replicas across a cluster to avoid correlated failures. It is done by grouping machines into a data center and a rack. DataStax Enterprise offers different kinds of snitches: DynamicSnitch, SimpleSnitch, EC2Snitch, Ec2MultipleRegionSnitch, GoogleCloudSnitch, etc.

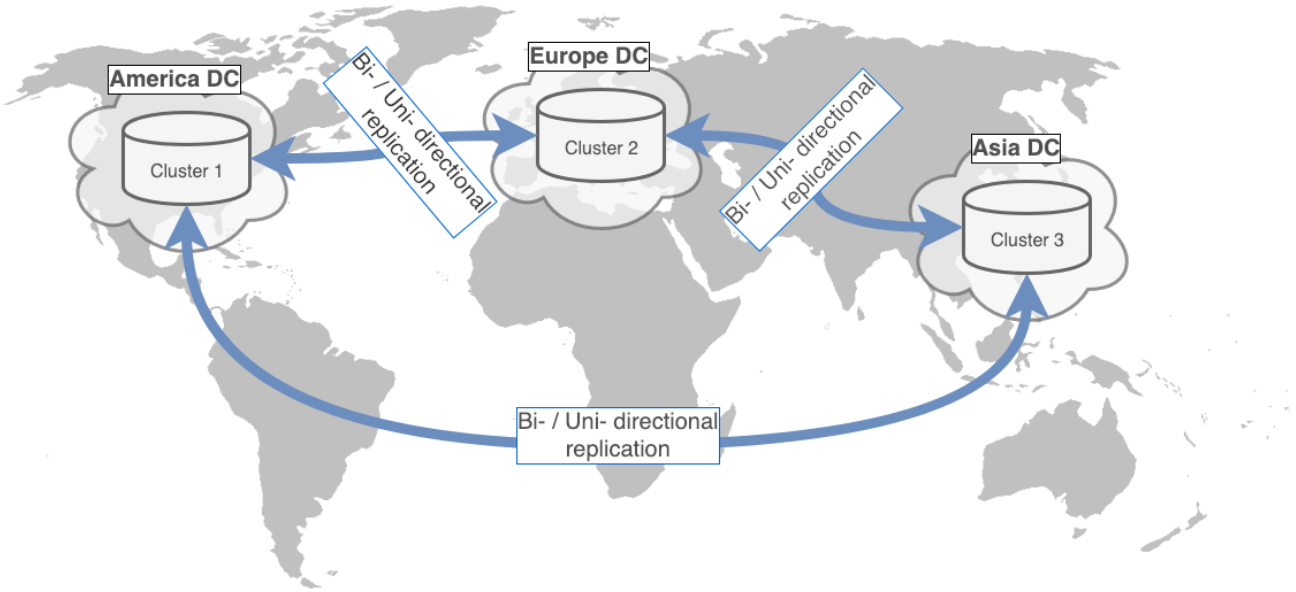


Figure 3.1.3.6 Multi-cluster replication by DataStax Enterprise (Cassandra)

A **replication factor** configures the number of nodes that will be responsible for handling vnodes. 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 as successful. If the tunable consistency level is lower than the replication factor, the record mutation is propagated to other replicas asynchronously.

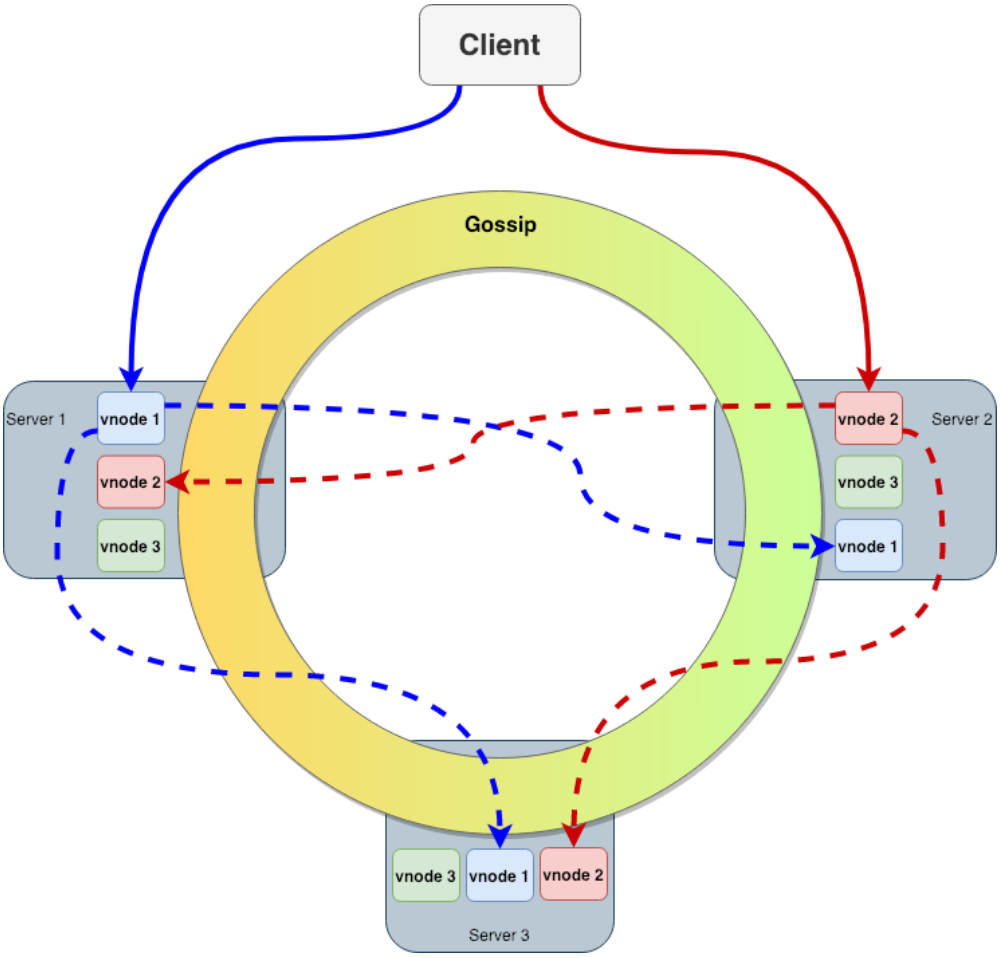


Figure 3.1.3.7 Intra-cluster replication in DataStax Enterprise (Cassandra)

DataStax Enterprise also offers DSE Advanced Replication. DSE Advanced replication enables configurable replication between a cluster, identifying sources, and a destination cluster with replication channels. Topologies such as *hub-and-spoke* or *mesh networks* can differently push or pull data depending on the operational needs.

DSE Advanced Replication can help to handle a scenario when a network of remote sensors has a poor connection to a centrally located storage and an analytics network. In this case, the remote edge clusters try to collect data and—based on the experience disconnections—send the updated data to the central hub when the connection is available. In addition, a central database cluster has an opportunity to send data to the remote cluster. Two channels should be designed for such a replication: *upstream* and *downstream* ones.

Summary

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

Table 3.1.3 Replication in the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	8	10

3.1.4 Consistency

The data consistency model specifies a contract between a programmer and a system by which the result of concurrent database operations will be predictable. Database consistency can also refer to 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 due to the asynchronous nature of replication between Couchbase Server clusters in remote data centers. XDCR provides two conflict-resolution strategies:

- The timestamp-based resolution, which means that “the most recent update wins.”
- The revision-based strategy, which means that “the most updates win.”

Global Secondary Indexes (GSI), view indexes, full-text search indexes, and analytics indexes are eventually consistent, as mutations are streamed and processed by the indexers asynchronously (which can lead to inconsistency). However, strong query consistency is still possible and achieved by applying a parameter on a per-query basis (`stale` on view queries and `scan_consistency` on N1QL 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 individual queries.

Couchbase supports ACID on single documents, but does not offer multi-document transaction support. Some workarounds can be used for supporting multi-document transaction. Future multi-document, distributed ACID transaction support is being planned.

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. For reads, MongoDB provides two capabilities for specifying the desired level of consistency: by specifying a maximum staleness value or using *read concern* to control the consistency. Non-default read preferences with default *write concern* ensure only eventual consistency, therefore changing read concern. Write concern should be tuned, as well. (The write concern option specifies how many nodes the primary will wait for before acknowledging a write operation.)

A user may 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 the loss of document updates, stale and dirty reads, as well as other problems. When an engineer changes a default value of the *read preference*, it may result in data inconsistency. To prevent this, one needs to customize write concern to a '*majority'* value and read concern to a '*linearizable'* value.

In addition, MongoDB introduces causal consistency, guaranteeing that every read operation within a client session will always see the previous write operation, regardless of which replica is serving the request.

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.

MongoDB applies write operations in their original order in batches using multiple threads. To provide consistency on batch operations, MongoDB blocks all read operations. As a result, secondary read queries can never return data that reflects a state which never existed on the primary.

When a 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. For situations that require atomicity for updates of multiple documents or consistency between reads of multiple documents, MongoDB uses multi-document ACID transactions for replica sets. The remaining features needed to deliver transactions across a sharded cluster are almost ready and announced to be delivered in version 4.2.

With multi-document transactions, no write operations are visible outside the transaction until it is committed. In other words, the multi-document transactions are atomic. For multi-document transactions, the '*snapshot'* read concern level is introduced.

DataStax Enterprise (Cassandra)

According to 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, a client application must use read and write consistency levels, so that the `(nodes_written + nodes_read)` value is bigger than `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 the `(nodes_written + nodes_read)` value is less or equals `number_of_replicas`. To resolve it, Cassandra uses the *manual repair* (full and increment) and automatic *read repair* or *node sync* process. During the process, replicas exchange data with each other, and the version of data with the most recent timestamp is stored on all the

replicas. One of the latest features of DataStax Enterprise (Cassandra) for synchronization is *nodesync*. It is an easy-to-use continuous background repair that has low overhead and provides consistent performance. The DSE Management Service supports a special service for working with *nodesync*—NodeSync Service.

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

The DSE Management Service provides a service to repair the run operations, which synchronizes the most up-to-date data across nodes and their replicas, including repairing any corrupted data encountered at the file system level. This service is called Repair Service, supporting three types of repairs:

- an incremental repair
- a subrange repair
- distributed subrange repairs

Summary

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

Table 3.1.4 Consistency of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
9	9	9

3.1.5 Availability

Availability refers to the capability of accessing a database cluster in case a single or multiple nodes go 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

The Query and Eventing Services are automatically load-balanced via the SDK. In the event of a query node failure, currently running queries will be affected and can be immediately retried (the SDK will transparently redirect to a remaining node). The Eventing Service will automatically ensure no loss of processing in the event of a node failure.

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 if the failed node is still within the cluster.

In terms of the CAP theorem, Couchbase Server is a CP system, favoring consistency and partition tolerance over availability. Here's the process.

- **A single cluster.** Within 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 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 node recovery. Couchbase offers Server Group Awareness that provides enhanced availability. It prevents a large-scale infrastructure failure through the definition of groups. All vBuckets can be organized in a group. Groups should be defined in accordance with the physical distribution of nodes. In this case, Group Failover is enabled—if all components of a group go down, replica vBuckets from another group will be automatically promoted to *active*. Data protection is optimal when groups have equal numbers of nodes. It allows for distributing vBuckets in such a manner that no replica vBucket will belong to the same group as an active vBucket. Server Group Awareness only applies to Data Service.
- **Two and more clusters.** XDCR offers two ways of replication: *unidirectional* replication and *bidirectional* replication. In the case of unidirectional replication, the level of availability will be close to the level of availability for a single data center. In the case of bidirectional replication, with XDCR, multiple clusters can act as write masters. This means the same document can be updated in two different locations (different clusters) before it is replicated. Used in different combinations, unidirectional and bidirectional replications can support complex topologies. For example, it can be a ring, chain, mesh, one-to-many, many-to-one, or some other topology. From the client perspective, the Couchbase SDK provides multi-cluster awareness, enabling automatic failover and load balancing between clusters.

Currently, N1QL queries do not have the ability to read from replicas, so they will experience failures if reading from the Data Service is required. Thus, a single Data Service node failure can cause unavailability of a range of N1QL queries. This can be mitigated by using covering indexes that can be partitioned and stored separately from data. Replica reads through N1QL are on the roadmap for an upcoming version.

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, but the read queries can continue to be served by the replica set if such queries are configured to run on secondaries. By enabling read operations on secondaries, data consistency can be managed using the *write* and *read* concern configuration. The failover process itself does not require manual intervention.

Configuration servers are also deployed as a replica set to tolerate failures of different types of cluster nodes. Special attention should be paid to the deployment design of configuration servers and mongooses. Otherwise, in case a configuration server replica set loses its primary, and no new primary can be elected due to misconfiguration, the whole cluster enters the read-only mode. If all the configuration servers or mongooses are unavailable, the cluster becomes inoperable.

DataStax Enterprise (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. If we lose a number of nodes that equals `(replication_factor - 1)`, the cluster data will still be available. Multi-datacenter replication ensures high availability in case of the entire data center failure.

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 and ability to isolate workloads. However, until an *active vBucket* is unavailable, write operations and some of the N1QL queries might not be executable.

Table 3.1.5 Availability of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
8	6	10

3.1.6 Server and network fault tolerance

Fault tolerance is the 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 a CP system: it provides consistency and partition tolerance. Data is horizontally partitioned, replicated, and distributed across 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.

There are two basic types of failover: *graceful* and *hard*. Graceful failover must be initiated manually and only applies to the Data Service. Hard failover can be manually initiated and can also be initiated automatically by Couchbase Server. Failure detection in Couchbase Server is based on several different signals:

- Heartbeats from nodes are analyzed by the Couchbase orchestrator before marking a node as unavailable.
- Monitoring of replication traffic between data nodes.
- Monitoring of a disk/write failure. If a node has a significant rate of failure, it is auto-failed over.

The minimum delay is necessary to prevent the transient network issues or a temporary node freeze from triggering the failover. Couchbase Server takes 6–7 seconds to process the failover. So, if there is a network partition between several nodes in a cluster, the automatic failover is not triggered. However, the remaining subclusters 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 Server 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, Index, and Analytics services can be failed over without any recovery delay, because they are ready-only and can be load-balanced more gracefully. The Cluster Manager provides load balancing out of the box.

Due to an independent architecture of Couchbase Server, it can support fault isolation. It means that failure or degradation of any service/component does not impact the behavior or availability of a node or cluster as a whole.

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 for data shards and configuration servers. 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, but can continue to serve read queries if such queries are configured to run on secondaries. If the primary node was unavailable, one of the remaining secondaries calls for an election to select a new primary and automatically resume normal operations.

Each member of a replica set has priority, which indicates the relative eligibility of a member to become a primary. A lower-priority instance can be elected as primary for brief periods, even if a higher-priority secondary is available. Replica set members continue to call elections until the highest-priority member available becomes primary.

While replica sets provide basic protection against a single-instance failure, replica sets which members are all located in a single data center are susceptible to data-center failures. Depending on the distribution of members across two or more data centers, if a data center goes down, the replica set can become read-only or remain writeable. To protect data in case of a data-center failure, it is a good practice to keep at least one member in an alternative data center.

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 a replica set, the primary becomes a secondary.

To tolerate cluster-node failures, nodes and configuration servers should be deployed as replica sets. However, if the configuration 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 configuration servers or mongooses are unavailable, the cluster becomes inoperable.

Fault tolerance can be increased by adding additional members (hidden or delayed); the members can provide support for dedicated functions, such as backups or reporting.

Replica set members send heartbeats (pings) to each other every two seconds. If a heartbeat does not return within 10 seconds (configurable with the `heartbeatTimeoutSecs` option), the other members mark the delinquent member as inaccessible. The time limit in milliseconds for detecting when the replica set's primary is unreachable is 10 seconds and configurable with the `electionTimeoutMillis` option. Lower values result in faster failover, but increase sensitivity to the primary node or network slowness/spottiness, while higher values result in slower failovers, but decrease sensitivity to the primary node or network slowness/spottiness.

MongoDB combines fast failover times with retryable writes. With this, MongoDB provides automatic support for retrying writes that have failed due to transient system errors, such as network failures or primary elections. This significantly simplifies application code. The database drivers can detect the loss of the primary and automatically retry certain write operations a single time, providing additional built-in handling of automatic failovers and elections.

With journaling enabled, it becomes possible to guarantee that a *mongod* will be able to recover its data files and keep them in a valid state following a crash. Write-ahead logging to an on-disk journal guarantees that MongoDB can quickly recover write operations written to the journal, but not written to data files in cases where a mongod is terminated due to a crash or some other serious failure.

DataStax Enterprise (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 reduced throughput. The missing write requests will be replayed on the failed node after it rejoins the cluster—by a process called a hinted handoff. The process involves a replica that stores incoming requests received during three hours (by default), while the other replica is offline, and then shares these requests with the second replica after an offline replica comes back.

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

Summary

Cassandra tolerates network partition or multiple server failures without any significant timeout. Couchbase Server and MongoDB require a certain amount of time for recovery. In case of MongoDB, a replica set turns to the read-only mode if a primary cannot detect the majority of the secondaries due to the network partition. If MongoDB loses all the configuration servers or router processes, it will not be possible to read and write data.

Table 3.1.6 Server and network fault tolerance of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
9	7	10

3.2 Administration

3.2.1 Configuration management

Configuration management embodies the means of keeping cluster node configuration in a coherent state during system bootstrap and at run time. Easy configuration management can dramatically improve user experience with a product.

Couchbase Server

A Couchbase Server cluster can be configured from an arbitrary node using a web UI or CLI, both based upon a supported REST API. Multi-dimensional scaling—independent scaling of the Data, Query, Analytics, Search, and Index services—can be done through the web UI just by turning the corresponding services on and off while adding a new node to a cluster. Alternatively, you can conduct multi-dimensional 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.

MongoDB

The bootstrapping properties for a MongoDB cluster can be specified with YAML configuration files or a CLI. The MongoDB shell is used to manage clusters and databases at run time. Additionally, Cloud Manager, Atlas (hosted services), and Ops Manager provide user interfaces and APIs for cluster deployment automation, configuration management, monitoring, and other capabilities.

DataStax Enterprise (Cassandra)

Several property files are used for configuring a Cassandra cluster—responsible for configuring a network, security, main memory and persistent storage, logging, backups, compression, etc. The nodetool is able to update the configuration properties at run time for a single node only. DSE OpsCenter enables you to configure and apply changes to a particular node or to the whole cluster using a web UI. DataStax Search and Analytics services also can be managed using DSE OpsCenter. In addition, DataStax Enterprise provides the OpsCenter REST API for managing a cluster. Due to the strong cohesion between the number of replicas and consistency levels in Cassandra, customer applications should be aware of the database deployment details (a replication factor in particular) to operate properly across all the development environments.

Summary

All the three products provide a web UI and REST API to configure particular nodes or the whole cluster.

Table 3.2.1 Configuration management of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	10	10

3.2.2 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 provides a number of tools for backup:

- cbbackupmgr
- cbbackup and cbrestore

Cbbackupmgr 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 configuration.

The *cbbackup* and *cbrestore* tools perform incremental backup. Incremental backup ensures a copy of only modified data in the database. This makes incremental backup and restore more efficient for a larger data set. Couchbase Server offers three types of incremental backup:

- differential
- cumulative
- a combination of the two above-mentioned types

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. One can also use a combination of both differential and cumulative incremental backups.

Cbbackupmgr supports three types of backup strategies: *periodic merge*, *full / incremental approach*, and *full backup only.* The *periodic merge* strategy creates and later merges incremental backups after a specified time period using a single backup repository. The *full / incremental approach* creates a new backup repository periodically and backs up the entire cluster again unlike *periodic merge.* The *full backup only* is useful for small clusters exclusively and puts the most strain on a cluster if compared to other strategies.

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.

MongoDB

MongoDB supports several methods to enable backups and restore data. Atlas, Cloud Manager, and Ops Manager allow for continuously backing up and making snapshots. Continuous backing up creates snapshots of cluster data at specified intervals and can also offer 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 underlying data files (file system snapshots). This method is provided by an operating system's volume manager and is not specific to MongoDB, but is in common

use. File system snapshots are created quickly and work reliably, though you have to make more configuration changes outside MongoDB. The MongoDB backup and restore utilities (*mongodump* and *mongorestore*) can be problematic for sharded clusters and replica sets, but work well for small-scale deployments. Instructions for the backup and restore of sharded clusters can be found in the official documentation. It is possible to backup and restore data using the API methods available in Atlas and Ops and Cloud managers.

MongoDB provides a way to configure delayed replica set members. These members can make it possible to recover from unsuccessful application upgrades and operator errors, including dropped databases and collections.

DataStax Enterprise (Cassandra)

The immutability of the SSTable files enables Cassandra to make snapshots by hard-linking original files instead of copying. After a snapshot is complete, these files should be copied to some other location. You can use the nodetool to make and remove a snapshot on a particular node. 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 newly flushed SSTables. Commit log backups facilitate the SSTable backup data for further recovery to a particular point-in-time state.

The DataStax Management Service offers Backup Service through which one schedule an automatic backup or run a manual backup of Cassandra cluster data. Local and remote backups, point-in-time recovery, and restoring to a different cluster can be also performed by Backup Service. In addition, this service can store backup data on each node (on a server), in a local file system, or on a cloud-based storage service (such as Amazon S3).

Summary

All the three databases provide multiple options for backup. Each database has its own pros and cons. Unlike Cassandra and MongoDB, Couchbase Server is not able to create point-in-time backups. However, Couchbase Server does allow you to back up data on a remote cluster by means of XDCR. DataStax Enterprise (Cassandra) can store backup data out of the box both in a local file system and a cloud-based storage, such as Amazon S3.

Table 3.2.2 Backup in the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
9	9	9

3.2.3 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 option is to enable periodic incremental backups using either the *cbbackupmgr, cbbackup*, or *cbrestore* tools. If a periodic backup is set up and running, then, in case of a serious failure, you will be able to restore data from its copies. However, a 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 missing data partitions from a backup cluster using the *cbrecovery* tool. This tool allows for previewing a list of buckets that are no longer available in a cluster and recover missing buckets from a remote cluster.

Furthermore, for disaster recovery, one can use an additional cluster that can act as a hot standby. In this case, an additional cluster can take over load, as soon as a main cluster stops responding.

MongoDB

Continuous Atlas and Ops/Cloud Manager backups are ongoing processes that work in a similar fashion as replica set data synchronization. By enabling checkpoints, it is possible to permit point-in-time restores in between 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, but with a lower election priority. This ensures only nodes in the active data center will be primary, unless they are all unavailable due to a complete data-center outage. In cases when some members in a replica set are unavailable, it might be required to perform the procedure to reconfigure a replica set with unavailable members, for example, in a geographically distributed replica set, where no local group of members can reach a majority.

DataStax Enterprise (Cassandra)

One of the ways to handle a global disaster is to recover a cluster using backups. Cassandra provides snapshots, as well as incremental and commit log backups. In addition, scheduling regular or one-off backups of all or a specific keyspace in a cluster can be configured by the OpsCenter Backup Service. This can help you to recover data from the stored 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 another 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's web UI.

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. In MongoDB, some manual steps might be needed to reconfigure a replica set in case a majority of nodes are unavailable.

Table 3.2.3 Disaster recovery of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	9	10

3.2.4 Maintenance

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

Couchbase

You can maintain Couchbase Server through the Couchbase Web Console, a CLI, and a REST API. The Couchbase Web Console 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.

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 these 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 automatic failover. However, later on, it will require a manual rebalance operation trigger. In addition, multiple node failure can be resolved only by manual intervention.

Couchbase, Inc. provides maintenance for Couchbase Server in the cloud. Couchbase Managed Cloud (CMC) offers a wide range of functionality for eliminating the burden of administering your database environment. CMC can be used for deployment, provisioning, upgrade, performance and capacity planning, as well as life cycle and infrastructure management. CMC promotes financial flexibility, because it offers Couchbase Server “as a service” on any cloud. In addition, CMC supports cross-cloud replication, which means data can be replicated between different cloud providers. With this functionality, you can easily migrate from one cloud provider to another.

MongoDB

In MongoDB, all the maintenance tasks for a cluster can be performed in Atlas or via Ops and Cloud managers. All the solutions provide a UI and REST API for cluster management. Cluster maintenance can also be performed with mongo shell methods.

Under maintenance tasks, one can scale in and out without taking an application offline, perform topology changes, upgrade without any downtime, make scheduled point-in-time backups, specify security configuration, resize the oplog, etc.

DataStax Enterprise (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's web UI. The nodetool runs commands for a particular cluster node, while OpsCenter enables you to execute tasks for all the nodes in a cluster. Typical maintenance tasks, including Cassandra version update, are performed online without any downtime. vnodes are evenly redistributed among all the cluster nodes during these activities.

DSE Management services provide sufficient functionality for maintaining a cluster, including NodeSync, Repair, Best Practice, Capacity, and other services. All these tools eliminate the burden of managing your database environment.

DataStax, Inc. provides the maintenance for DataStax Enterprise (Cassandra) in the cloud. DataStax Managed Cloud Service offers a management console, provisioning, proactive intervention/alerting, backup and restore, upgrades, and log access. Currently, DataStax Management Cloud supports deployments only on Amazon Web Services or Microsoft Azure.

Summary

Couchbase Server offers a web console that is designed to perform the majority of maintenance tasks in a single place. DataStax Enterprise (Cassandra) and MongoDB do not provide such functionality out of the box, but have similar solutions that must be set up separately and require additional effort for deploying.

Table 3.2.4 Maintenance of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	8	9

3.2.5 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 should be triggered, either automatically by a cluster or manually by an administrator or a script. The failover operation immediately removes the node from the cluster and activates replicas for its data, which 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 operation is needed to return the cluster to a healthy and fully replicated state. Rebalancing 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 methods. The full recovery strategy completely cleans up the failed-over node and restores data from other cluster nodes. The delta recovery strategy reuses data on the failed-over node. It checks the existing data to identify the point when mutations stopped, and synchronization begins at this 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 this case, the delta recovery strategy defaults to a recovery.

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 calls for an election to select a new primary and automatically resume normal operations. 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. A full copy is only initiated if the rejoining node has become too stale during its unavailability. To sync the rejoined member by copying data files from another member, the data files must be sufficiently recent to allow the rejoined member to catch up with the oplog. The oplog size is configurable, and the sync source is automatically selected based on the heartbeat (ping) results. Though copying data files from its sync source allows for updating a rejoined node more quickly, the procedure implies more manual steps than performing a full copy.

Users might lose data as a result of a "rollback." To prevent it from happening, one can employ the *write concern* option, which specifies how many replica set members the primary will wait for before returning the acknowledgement to the issuing client. The default write concern implies that write operation has propagated to the standalone mongod or the primary in a replica set. The default write concern can be changed to *majority* not to lose updates, when the former primary member rejoins its replica set after a failover. As a result of the rejoin operation, the rejoined member that was previously the primary might become a secondary with a diverged oplog. This divergence is fixed by the rollback procedure that discards the updates acknowledged by the former primary, still never being replicated to the secondaries. A rollback does not occur if the write operations replicate to another member of the replica set before the primary steps down and if that member remains available and accessible to a majority of the replica set.

DataStax Enterprise (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 following repair process are provided by DataStax Enterprise:

- *Hinted Handoff.* The writes coordinator node preserves the data to be written as a set of hints. When the node comes back online, the coordinator hands off hints, so that the node can catch up with the required writes.
- *Read Repair.* In case the coordinator detects that a replica node has outdated data, the coordinator node sends it the most recent version.
- *Anti-Entropy Repair.* The Nodetool repair tool is used to ensure data consistency across a cluster.
- *NodeSync.* This is the process for continuously verifying data consistency across replicas in the background.

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 enables 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.

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. While Couchbase Server and DataStax Enterprise (Cassandra) provide functionality that automatically synchronizes data to a rejoined node, MongoDB requires manual steps to be done for syncing data to a rejoined members of a replica set.

Table 3.2.5 Recovery of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	9	10

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:

- A *web console* that provides a centralized view of all the 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* that exposes endpoints, providing cluster-, node-, and bucket-level statistics. Email alerts can be configured for the REST API in the same way as for the Web Console.
- The *cbstats* utility. 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.

The Couchbase REST API or Couchbase Server command-line tool can be used to monitor more than just a basic node status. These tools can provide information about *node warm-up status*, *current operations per second*, and *rebalance progress*.

Couchbase Server offers monitoring management. Gathering statistics through using different configurations, file locations, and methods can be tuned and customized specifically for your environment. The default Couchbase Server statistics collection is set to gather data every second, but this parameter can also be customized to fit your needs.

Monitoring and profiling N1QL, query service engines, and a corresponding system resource is useful for analyzing current operational performance and efficiency of the system. In general, such issues as query performance, resource bottlenecks, and overloading of services might exist in a database environment. In this case, Couchbase Server offers various monitoring details and statistics, which can prevent the aforementioned issues.

For example, the following statistics can be identified:

- the top ten slowest or fastest queries
- resources usage statistics of the query service or a particular query
- details about the active, completed, and prepared queries
- long queries running for more than two minutes

MongoDB

MongoDB (Community Edition) offers free cloud monitoring for standalones and replica sets. A user can monitor information about database deployment, including operation execution times, memory usage, CPU usage and operation counts.

Atlas and Cloud Manager provide a possibility to monitor cluster, replica set, and MongoDB processes metrics. The metrics, as well as alerts and events, can be accessible via Web UI or Atlas and Cloud Manager APIs. The Monitoring Agent is implied to be installed for Cloud Manager to enable the monitoring functionality.

MongoDB Ops Manager offers a Monitoring Agent that can be installed as a light-weight component on cluster nodes. This agent collects various cluster metrics and provides visualization functionality. Cluster metrics, events, and alerts can be accessible via a web UI or the API. GUI tools such as Charts (available in beta) and Compass allow for exploring, analyzing, and manipulating data.

Several complementary methods can be used for collecting data about the state of a running MongoDB cluster. A set of utilities distributed with MongoDB provides real-time reporting of 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, as well as reports these statistics on a per-collection basis.

The MongoDB 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. SNMP is available in the MongoDB Enterprise version.

Various third-party monitoring tools, such as Ganglia, mtop, Nagios, Munin (mongo-munin, mongomon, and munin-plugins Ubuntu PPA), and Wireshark support MongoDB either directly or through their plugins, as well as various Hosted (SaaS) Monitoring Tools.

DataStax Enterprise (Cassandra)

DataStax Enterprise (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 the DataStax Enterprise (Cassandra) distribution, facilitates monitoring and maintenance. For example, *nodetool tablestats*, *nodetool tablehistograms*, *nodetool netstats, nodetool tpstats*, *etc.* can provide statistics about a single or multiple tables. The *nodetool* uses the metrics-core library to make output more informative and understandable.

OpsCenter is a graphical web UI that enables you to monitor all the nodes in a cluster from a single place. DSE OpsCenter has a Performance Service and a Best Practice Service. Performance Service includes OpsCenter metrics with CQL-based diagnostic tables. Therefore, this service can help to analyze, tune, and optimize cluster performance. Best Practice Service periodically scans clusters to automatically detect issues that affect the cluster's health. The OpsCenter API provides an ability to get all the metrics by RESTful requests for programmatically performing the same set of operations as OpsCenter Web UI.

JConsole is a built-in Java graphical desktop user interface that consumes and displays the JMX metrics. It is also possible to pull Cassandra metrics through the JMX HTTP bridge. The following types of information can be collected:

- cluster statistics
- thread pool and read/write latency statistics
- table statistics
- compaction and compressing metrics
- endpoint metrics
- NodeSync metrics
- query statistics, etc.

One of the features that was introduced by DataStax Enterprise (Cassandra) in the current version is DSE Metrics Collectors. DSE Metrics Collectors aggregates metrics and integrates with above-state

monitoring solutions. DSE Metrics Collector is built upon collectd. Cassandra automatically sends metrics and events to DSE Metrics Collectors.

Summary

All the solutions under consideration allow you to perform monitoring using the command-line shell, a web UI, and a REST API. The quality and quantity of the collected statistics are up to the mark.

Table 3.2.6 The ease of monitoring the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	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 enables you to implement authentication by means of the Lightweight Directory Access Protocol (LDAP) or Pluggable Authentication Modules (PAMs).

Couchbase Server supports X.509 certificate-based authentication. It means that only approved users, machines, or endpoints are authenticated. Data, Query, Analytics, and Search Services provide access by this authentication mechanism.

For authorization, Couchbase Server comes with role-based access control (RBAC). It provides built-in roles that can be assigned per user or per bucket for each data access type (such as key-value, Query, 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.

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

Over-the-wire, client-server, and XDCR traffic encryption are achieved using transport layer security (TLS) in 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. Native intra-cluster encryption is scheduled for the next major release.

Couchbase Server also offers field-level encryption. The goal of field-level encryption is to protect sensitive user data.

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. All audit logs are written to a specific target file, which is periodically rotated.

Log redaction can be enabled for preventing a leak of sensitive data if access to the logs for troubleshooting production issues should be provided.

MongoDB

MongoDB supports the following authentication mechanisms: SCRAM-SHA, x.509, LDAP, and Kerberos. The API uses HTTP Digest Authentication. 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 predefined built-in roles.

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

MongoDB Atlas and Enterprise editions provide administrators with a system auditing facility that can record system events, such as user operations and connection events. This audit records permit forensic analysis and allows administrators to verify proper controls over the system access rights.

In MongoDB, it is possible to restrict the contents of the documents based on the information stored in the documents themselves by implementing Field Level Redaction.

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.

DataStax Enterprise (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.

For Authentication, Cassandra supports user validation with the following methods: *internal* (credentials stored in the internal database), *LDAP*, and *Kerberos.* DataStax Enterprise comes with an *internal database* and *LDAP* for authorization.

DataStax Advanced Security is a set of features to protect DataStax Enterprise databases. DataStax Enterprise Security can integrate with different kinds of technologies, such as Active Directory (AD), Lightweight Directory Access Protocol (LDAP), Kerberos, Public Key Infrastructure (PKI), and Key Management Interoperability Protocol (KMIP), etc.

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 Enterprise (Cassandra) enables you to encrypt

such data at-rest as tables, indexes, commit logs, and property files. Auditing provided by Cassandra grants system access control.

Cassandra can capture database activity to a log file or a table. Each node stores the events locally. Recording audit logs to a unified table has advantages compared to a log file, allowing collection of audit logs from multiple nodes.

Summary

All the three solutions provide RBAC, include the ability to integrate major authentication and authorization mechanisms, such as LDAP, allowing you to encrypt data at-rest, enable you to secure intra-cluster and cross-datacenter traffic, and perform audits. In addition, MongoDB supports STIG and Field Level Redaction that allows for restricting contents reading of the documents based on information stored in the documents themselves.

Table 3.2.7 Security of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
9	10	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 saves data as items. Items consist of a *key* and a *value*. Each key is unique within its buckets, and values can be either binary or a JSON document. The maximum size of a value is 20 MiB. A key has to satisfy the following parameters:

- must be a UTF-8 string with no space
- must not exceed 250 bytes in size
- must be unique within its buckets

Furthermore, Couchbase Server stores metadata for each item.

All items are saved in *buckets*. Different storage engines are used depending on a selected bucket type. Couchbase Server supports three types of buckets:

- Couchbase
- Ephemeral
- Memcached

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

Couchbase server provides different kind of *indexes* for enhancing performance of query and search operations. Currently, Couchbase Server has the following indexes:

- primary (provided by the Index Service)
- global secondary (provided by the Index Service)
- full-text (provided by the Search Service)
- view (supports Couchbase Views)

A primary index is based on the unique key of each item in a specified bucket. Global secondary indexes (GSIs) are used to retrieve data via the SQL-like syntax (N1QL). GSIs have two storage modes: standard and in-memory. The in-memory mode is implemented through a lock-free skip list and requires enough RAM to fit the entire index. If RAM is exhausted, the index will stop processing new updates, but will be still available to serve queries. The standard mode leverages the ForestDB storage engine to store the B-Tree index and keeps the optimal working set of data in the buffer. It allows an index to be larger than the RAM available at the potential cost of performance via disk access. Full-text indexes are a special-purpose index for the Search Service which provides a wide range of possible text-searches. Views extract specific attributes, aggregates, and any other type of information from documents by the user-defined MapReduce functions for querying purposes.

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 embedded documents and arrays within other documents and arrays. BSON-document size limit is 16 MB. Similar as files are split into chunks in HDFS, GridFS can be used for storing and retrieving files that exceed BSON-document size limit or for storing any files for which access is needed without having to load the entire file into memory.

In MongoDB, each document stored in a collection has a unique *_id* field that acts as a primary key. MongoDB driver will automatically generate the *_id* field and a value for it, in case the *_id* field is not specified for the document to be inserted in the database.

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

It is possible to configure MongoDB to use one of the following storage engines on a per-node basis: WiredTiger (default) and in-memory. 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 and compression for all collections and indexes. WiredTiger supports the LSM table types, and B-tree data structure is used within MongoDB for its indexes. The in-memory storage engine is non-persistent and does not write application and system data to a persistent storage. 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.

DataStax Enterprise (Cassandra)

Cassandra is a partitioned row store. A row consists of columns, which are organized into tables, and tables are grouped in a keyspace. Each table must have a unique primary key. A *primary key* consists of a *partition key* and a *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.

To query by columns, which are not part of a primary key, Cassandra uses *secondary indexes* and a *materialized view.* Secondary indexes and materialized views are implemented as regular tables with their own primary keys, which are kept in synchronization with the original tables. Secondary indexes are processed locally. Each index partition is stored on the same node as its original data partition. In addition, SSTable Attached Secondary Indexes (SASI) can be used for improving performance of secondary indexes. SASI were an improvement open-sourced (contributed by Apple) to the Cassandra community. Materialized views are partitioned independently from the original tables. A materialized view provides fast lookup of data using normal read path, but it does not have the same write path, because the database performs an additional read-before-write operation to update each materialized view. In general, secondary indexes are built faster during writes, but they are less efficient during reads than materialized views.

DataStax Enterprise (Cassandra) has not typical way of data modeling. Data modeling is a process that involves identifying the entities and the relationships between entities. Data modeling in Cassandra uses a query-driven approach. It means, specific queries are the key to organizing a data model. In this case, all tables should be created only after all possible queries were developed. In general, the process of data modeling does exactly the opposite.

When a write occurs, it is first stored to a *commit log* on a disk for the recovering purpose and in a memory data structure, which is called a *memtable*. 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.

DataStax Enterprise (Cassandra) offers *DSE In-Memory* that provides lightning-fast performance for read-intensive situations. Using DSE In-Memory, the database as a whole or parts of it reside fully in RAM. It is not suitable for write-intensive data or systematically growing datasets that might exceed the RAM capacity on the nodes.

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, as well as utilize disk space and memory in accordance with the particular data structures and formats. Couchbase Server and MongoDB are suitable solutions for storing the unstructured JSON documents, which 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, as well as requires a user 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 3.3.1 Data structure and the format of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	10	8

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 data in Couchbase Server. The key-value request is sent directly to a proper node, holding the target document, and does not require any index scanning.

Couchbase Servers relates to a document-oriented database, therefore, data is stored as a JSON document. For querying, Couchbase Server supports a query language called N1QL—a SQL-like query language with a set of specific extensions for addressing JSON data. This language features data query, data manipulation, and data definition capabilities, as well as role-based access control (RBAC). In addition, N1QL provides tools to manage query performance as prepared statements, allowing to skip the parsing and execution planning phases for repeated queries. N1QL also offers an explain plan option to investigate query execution flow and timings. In fact, the N1QL language is a commercial implementation of SQL++.

Here are some major advantages of N1QL:

- *Similarity to SQL.* N1QL incorporates a sophisticated `JOIN` capability to query over multiple documents, as well as some other SQL-like features. This is the reason why it is easy for developers to learn the language.
- *Declarative.* With N1QL, you are able to express what you want to accomplish, and Couchbase Server will figure out the most optimal way of accessing and processing this data to get the desired result.
- *Expressive power.* Based on SQL++, N1QL is a highly composable and expressive query language.
- *Tunable consistency.* The level of consistency is configurable for each query.

Global Secondary Indexes help to optimize different queries, increase query throughput, and decrease query latency. Couchbase Server offers a lot of indexing types:

- *Primary.* The primary index is simply an index on the document key on the entire bucket.
- *Named primary.* This index allows for having multiple primary indexes in the system.
- *Secondary.* This index can use any key within the document. The key can be presented in one of these types: as a scalar, an object, or an array.
- *Composite secondary.* This index is used for optimizing queries with multiple filters.

- *Functional.* The index provides the query engine with possibility to make more efficient search in case of a sensitive query.
- *Array.* This index is an expression on the array to clearly reference only the elements that need to be indexed.
- *Partial.* Due to this index, the type of field will be considered in queries.
- *Duplicate.* Couchbase Server offers to create duplicate indexes with distinct names for query optimization. During query execution, these indexes are used in a round-robin fashion to distribute the load.

As with SQL, it is possible to use 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.

The N1QL language is continuously developed by Couchbase Inc. In the recent versions, Couchbase Server supports different kinds of useful features:

- ANSI JOINs. With this option, it is possible to execute a join operation by any document key or any other field in a document. Note that for `ANSI JOIN` in any document field, an index has to be created for this field. Couchbase Server uses two approaches to execute `ANSI JOIN`. Nested Loop and HASH JOINs. Nested Loop is a default approach for `ANSI JOIN`. In case a field is not indexed, Nested Loop might take a very long time to run a query. Here comes in a HASH JOIN, which has two types: a BUILD and a PROBE. The BUILD type is used to create an in-memory hash table on the left, and PROBE is used to create an in-memory hash table on the right. The choice between PROBE and BUILD depends on the size of sets. Typically, the smaller of the two types should be created in-memory.
- Aggregate pushdowns. A query planner requests an indexer to perform grouping, aggregation, and scanning of a covering index. The indexer performs grouping, `COUNT`, `SUM`, `MIN`, `MAX`, `AVG`, and other operations on the fly. As a result, you can minimize the time spent on data transfer and disk I/O operations.

HASH JOIN and *Indexer grouping and aggregation* are supported only in the Enterprise Edition.

MongoDB

MongoDB provides its own API-based query language, comprising CRUD operations and aggregation commands, including the aggregation pipeline and map-reduce frameworks, as well as plenty of operators and modifiers that can be used for data queries.

MongoDB features a JavaScript interpreter, which allows for using the power of the JavaScript language for writing flexible scripts, employing callback functions, objects, lambdas, anonymous function, handling errors, etc. In case the *server-side execution* of the JavaScript option is enabled for a server, it is possible to execute JavaScript code directly, by specifying a .js file to the mongo shell. Interactions with MongoDB can also be done with the help of an idiomatic driver in the language of the interacting application.

In the aggregation framework, a query is a multi-stage pipeline, where each stage defines a transformation applied to the set of documents produced by the previous stage. Each pipeline stage may either produce zero, single, or multiple output documents per input document. Some stages take

a pipeline expression as an operand that specifies a document transformation. MongoDB provides a large set of stages and expressions out of the box that cover most of the SQL functionality.

With the map-reduce framework, documents are transferred through two stages: map and reduce. MongoDB applies the map phase to each input document (i.e., the documents in the collection that match the query condition). The map function emits key-value pairs. For those keys that have multiple values, MongoDB applies the reduce phase, which collects and condenses the aggregated data. All map-reduce functions in MongoDB are JavaScript and run within the mongod process. Sorting and limiting documents can be performed prior to the map stage.

Though MongoDB query language does not support joins, the `$lookup` operator added to the aggregation pipeline performs a left outer join to an unsharded collection. As for sharded collections—the MapReduce functionality can be used in order to combine data.

In MongoDB, an index supports a query when the index contains all the fields scanned by the query. The query scans the index, not the collection. Sort operations can obtain the sort order by retrieving documents based on the ordering in an index. The best indexes for an application must take a number of factors into account, including the types of queries to be executed, the ratio of reads to writes, and the amount of free memory in the system.

DataStax Enterprise (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), as well as 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. DataStax, Inc. offers DSE Performance Service for managing query performance. You can also use *nodetool* available via an open-source for such kind of things.

DataStax Enterprise (Cassandra) supports secondary indexes. *Secondary indexes* provide access to data using a column rather than a partition key. Unfortunately, *secondary indexes* do not show good performance on large volumes of data. So, it is recommended to index a column with low cardinality of a few values. In addition, Cassandra provides SSTable attached secondary indexes (SASI), which are significantly less resource-intensive, employing less memory, disk, and CPU than general *secondary indexes.* Under the hood of SASI, there are memory-mapped B+ trees. Currently, SASI indexes in DataStax Enterprise are experimental.

Another way to get access to data using a column rather than a partition key is materialized view—a table built from data in another table with a new primary key and new properties. Performance of read operations from a *materialized view* is the same as normal table, however, materialized view does not have the same write performance as a normal table, because the database performs an additional read-before-write operation to update each materialized view.

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 in comparison to CQL. MongoDB provides its own query API that covers most of the SQL features and enables users to employ the power of the JavaScript language for writing flexible scripts, but requires developers to learn the specific coding idiosyncrasies. CQL allows to perform filtering, ordering, and grouping operations only in 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.

In contrast to Couchbase, MongoDB has some operational restrictions in sharded cluster, while Casandra has quite a limited query language due to its wide-column oriented structure.

Table 3.3.2 A query language of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	7	6

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

Couchbase Server provides the Search Service for full-text search. Couchbase Server supports multidimensional scaling, which allows for scaling the Search Service independently from other services employed by the applications that require high search performance.

The Full-Text Search (FTS) functionality enables text indexing and analysis by incorporating the Bleve search library in Couchbase Server. Bleve is an open-source search and indexing library written in Go. 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 prebuilt analyzers out of the box. Currently, Full-Text Search supports 20 languages. Search results are ordered by relevance using the TF-IDF scoring and results can be set using boosting rules.

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 FTS nodes. If one of these nodes fails, FTS returns a list of partial results and notifies about the error. In addition, FTS supports role-based access control. This allows assigning of the role of *FTS Admin* to the users who are going to be creating an indexes. The role of *FTS Searcher* should be assigned to the users who are going to perform search.

In the next major version, Full-Text Search indexes will be integrated into N1QL directly as well.

MongoDB

MongoDB supports queries that perform text search of the string content using a text index. A collection can have a single text index maximum, but the index can include multiple fields, which value is a string or an array of string elements. Prior to performing text queries, you need to create an index for the target collection and specify the document fields that will be indexed.

For a text index built on multiple fields, a weight for an indexed field can be specified to denote the significance of the field relative to the other indexed fields in terms of the text search score. Wildcard Text Indexes can be used for highly unstructured data if it is unclear which fields to include in the text index or for ad-hoc querying.

Various languages are supported for text search. Text indexes drop language-specific stop words and use simple language-specific suffix stemming.

The text search is available in the aggregation pipeline via the use of the `$text` query operator in the `$match` stage.

DataStax Enterprise (Cassandra)

DataStax, Inc. offers a special service for full-text search—DataStax Enterprise Search, which provides an integration of Cassandra with Apache Lucene and Apache Solr. Apache Lucene is a search engine framework that adds search capabilities to an application via the Java API. Apache Solr is a standalone web application that uses Lucene as its core and ships additional enterprise features. Both Apache Lucene and Apache Solr are closely integrated and run in a single JVM.

Data is asynchronously indexed in Apache 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 nodes of DataStax Enterprise Search service are strongly fault-tolerant with no single point of failure, highly available, and linearly scalable as they use Cassandra under the hood.

Two indexing modes are supported by DataStax Enterprise Search. There is near-real-time (NRT) and live indexing, also called real-time (RT). NRT is the default type of indexing based on Apache Solr and Apache Lucene. RT allows for searching directly against the Lucene RAM buffer, which provides earlier visibility to the newly indexed data. At the same time, larger RAM buffer and more memory usage should be employed by RT, otherwise, performance of RT is equivalent to the performance of NRT.

Summary

Both MongoDB and Couchbase Server deliver full-text search capabilities out of the box. While full-text search is available via the aggregation pipeline in MongoDB, Couchbase Server has a dedicated Search Service. In contrast to MongoDB, Couchbase Server allows for scaling the Search Service independently from other services employed by the applications that require high search performance. DataStax Enterprise Search provides integration with Apache Lucene and Apache Solr—one of the most powerful combinations for text indexing and search. DataStax Enterprise (Cassandra) makes use of the DSE Solo approach to enable highly scalable search that does not

affect other database operations. MongoDB provides a possibility of integration with Elasticsearch and Solr.

Table 3.3.3 Full-text search capabilities of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
9	7	9

3.3.4 Analytics

Analytics refers to the capability of a database to analyze large volumes of data.

Couchbase Server

Couchbase Server provides the Analytics Service that allows to manage and run complex analytics queries. The Analytics Service is based on the massively parallel processing (MPP) query engine. MPP is capable of executing queries in parallel on multiple servers and leveraging all the cores on each server as needed to process large volumes of data efficiently. Due to multidimensional scalability, Couchbase Server can run complex and resource-intensive analytics queries with no impact on the query latency and throughput of the operational system.

The Analytics Service uses *N1QL for Analytics*, which supports *join*, *set*, *aggregation*, *grouping*, and other operators. In addition, it allows for manipulating nested objects and arrays in JSON data.

A lot of BI tools can be integrated with the Analytics Service. Couchbase Servers offers the CData drivers (ODBC/JDBC) for integration with any SQL-based BI tool such as Tableau and native integration is available via Knowi.

For deploying the Analytics Service, the Data Service must be running on at least one of cluster nodes for each service. The Analytics Service is responsible for processing massive arrays of data, therefore, it should be run alone, on its own cluster node, with no other Couchbase service running on this node.

For further data analysis, users may also employ the Couchbase Spark Connector.

MongoDB

MongoDB supports multiple data processing pipelines via ETL tools, custom code, and scripts. A rich collection of drivers and a powerful aggregation framework, which includes an aggregation pipeline and the map-reduce functionality, enable users to create custom solutions, while the support of server-side JavaScript allows for triggering custom business logic.

The database can be integrated with the leading data integration tools, such as Alteryx, Talend, Pentaho, Knowi, etc. Custom solutions can be integrated with various charting libraries, such as d3, Vega, Vega.lite, chart.js, etc.

With MongoDB Spark Connector, users have access to all the Spark libraries to use with MongoDB data sets.

MongoDB BI connector enables users to exploit MongoDB as a data source for BI and analytics platforms, as well as create visualizations and dashboards that help to extract the insights and hidden value in multi-structured data.

MongoDB Compass has a visual schema analyzer and an aggregation pipeline builder. The GUI application is designed for in-depth analysis and visualization of MongoDB data and a collections' schema.

MongoDB Charts (beta) provides data visualization, highlighting correlations between variables and making it easy to discern patterns and trends within a data set.

DataStax Enterprise (Cassandra)

DataStax, Inc. offers a special service for analytics—DSE Analytics, which supports real-time and batch operational analytics. With DSE Analytics, large volumes of data can be easily processed and analyzed.

DSE Analytics is based on Apache Spark and provides the following features:

- *No single point of failure.* Thanks to the peer-to-peer architecture, any analytics node can be responsible for Spark Master.
- *Spark Master Management.* DSE Analytics provides automatic Spark Master management.
- *Analytics.* A Spark job can be run directly against data in the database. Real-time and analytics workloads can be performed without affecting each other.
- *DataStax Enterprise file system (DSEFS)* is a fault-tolerant, general-purpose, distributed file system by DataStax, Inc. DSE Analytics has a good integration with DSEFS and may use it for data ingestion, data staging, and state management of Spark streaming.
- *DSE Analytics Solo* is an approach to deploying DSE Analytics in a way that does not affect the database performance.
- *Integrated security.* DSE Analytics can use all the advanced security features of DataStax Enterprise (Cassandra).

Cassandra provides a possibility to integrate DSE Analytics with DataStax Enterprise Search. The integration enables finer-grained control over the types of queries that are used in analytics workloads, as well as improves performance by reducing the amount of data processed.

Summary

With Analytics Service, Couchbase Server can run complex and resource-intensive analytics queries with no impact on the query latency and throughput of the operational system. Couchbase Analytics Queries are based on the SQL++ language. DataStax Enterprise (Cassandra) offers to use Apache Spark out of the box for building analytics operations. Using DSE Solo, you can run analytics queries with no impact on database operations. MongoDB can be integrated with different tools that come useful in data analytics. MongoDB cloud offering provides tools for exploring data statistics and visualization.

Table 3.3.4 Analytics capabilities of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
9	7	9

3.3.5 Eventing and trigger capabilities

Eventing refers to the capability of a database to support real-time data processing.

Couchbase Server

Couchbase Server offers the Eventing Service, which makes it possible to enjoy the following functionality:

- set alerts in a document when a preconfigured threshold is breached
- monitor specific parameters in a document
- propagate changes to other systems
- enrich a document in real time
- cascade deletes to avoid an orphaned document

Couchbase Functions are used for writing *server-side functions*, which are automatically triggered in Event Service using the *Event-Condition-Action* model. Couchbase Functions are written in JavaScript. Under the hood, Couchbase Functions inherit support for most ECMAScript constructs by using Google V8 as an execution container for JavaScript code. Often, developers need to debug and log their own code, and Couchbase Server provides such a possibility. A visual debugger helps developers to debug the code before deploying it in the environment. For logging, the Eventing Service creates two different types of logs: *system* and *application* ones. *System logs* contain an error that relates to "redactable" user data. *Application logs* contain errors that relate to function activities. Couchbase server can also calculate and fetch event statistics from an eventing node using the Web Console.

The Eventing Service offers to use three event handlers:

1) *OnUpdate handler* is called when a document is created or updated.
2) *OnDelete handler* is called when document is deleted
3) *Timer handler* is called in reference to wall-clock events.

It is not recommended to deploy the Eventing Service with other services, because it is a compute-oriented service. For this reason, the Eventing Service should be deployed on core-heavy instances standalone.

MongoDB

MongoDB provides change streams that allow applications to access real-time data changes without the complexity and risk of tailing the oplog. Applications can use *change streams* to subscribe to all data changes on a single collection, a database, or an entire deployment.

To get a collection, a database or a deployment changes in an application, a change stream should be opened. Then, it is possible to iterate over the cursor to retrieve the change stream documents. While the connection to the MongoDB deployment remains open, the cursor remains open. It is possible to control change stream output by providing an array of one or more pipeline stages when configuring the change stream.

Change streams only notify about data changes that have persisted to a majority of data-bearing members in the replica set. This ensures that notifications are triggered only by majority-committed changes that are durable in failure scenarios. Change streams can work with multi-document transactions, in case data is changed with the transaction, the change event document includes the txnNumber and the lsid.

MongoDB offers Stitch, which provides triggers—based on *change streams*—that enable processing and reacting to events in the database as they occur. It becomes possible through linking the triggers with a specific Stitch function. One can automatically respond to changes in a MongoDB collection with a database trigger or execute additional server-side logic when a user is created, authenticated, or deleted via an authentication trigger.

Using MongoDB Stitch *database triggers*, users are able to implement complex data interactions, including updating information in a document, when a related document changes, or interacting with a service, when a new document is inserted.

Stitch *authentication triggers* allow for executing a server-side logic—whenever a user interacts with an authentication provider—as well as implementing advanced user management, including storing new user data in MongoDB, maintaining data integrity when a user is deleted, or calling a service with users' information when they log in.

DataStax Enterprise (Cassandra)

Cassandra supports triggers. The logic of a trigger can be written in Java. The code for the *trigger* should be located in the *lib/triggers* subdirectory of the Cassandra installation directory. *Triggers* load during the cluster startup and exist on each node of a cluster. A *trigger* attaches to the table/column family. A *trigger* ensures the atomicity of the transaction, because it calls before the requested DML statement occurs. DataStax Enterprise marks a *trigger* as deprecated functionality.

Summary

Couchbase Server offers the Eventing Service capable of monitoring specific fields, setting alerts, enriching documents, cascade deleting, etc. However, Couchbase Function can be written only in JavaScript. DataStax Enterprise marks *triggers as deprecated*, but open-source Cassandra is still supporting *triggers*. MongoDB introduces change streams that help to access real-time changes on a collection, a database, or a deployment, as well as helps to respond to the changes. MongoDB Stitch, offered only with Atlas, extends *change streams* functionality providing database and authentication *triggers* that can listen to application events and execute a Stitch function as they occur. In contrast to MongoDB, Couchbase Server follows the multi-dimensional paradigm, which allows for scaling the Eventing Service independently from other services.

Table 3.3.5 The event and trigger capabilities of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
9	8	5

3.3.6 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 for 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).

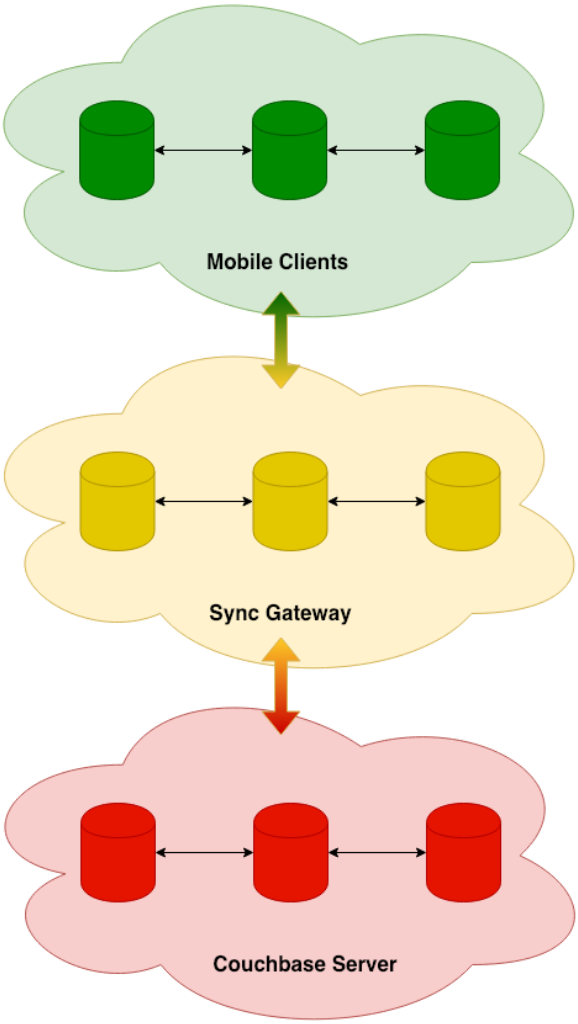


Figure 3.3.6.1 The mobile stack of Couchbase Server

Couchbase Lite manages, stores, queries, full-text searches 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 enables a mobile device to accept connections from other devices running Couchbase Lite and exchange data with them. Thanks to the support of the P2P architecture, Couchbase Lite can synchronize data without any centralized entity. In addition, a Couchbase Lite user can modify data offline, and this data will be synchronized with the Couchbase Server or a P2P network via Sync Gateway once online. For security, Couchbase Lite provides disk encryption with AES-256 to address data security and SSL/TLS. In case of data recover, Couchbase Lite has a possibility to back up and synchronize with an external store on the same device. Couchbase Lite can be deployed on a variety of platforms, such as iOS, Android, and Windows.

Couchbase Sync Gateway is responsible for synchronization, authentication, authorization, access control, and data routing between Couchbase Lite and Couchbase Server. Couchbase Sync Gateway provides wide flexibility for deploying. It can be deployed on cloud providers (Amazon Web Services, Microsoft Azure, or Google Cloud Platform), on premises, or in a private cloud. Couchbase Sync Gateway uses a time-tested synchronization/replication protocol that has evolved over the years to meet security, performance, and scalability demands of business-critical applications.

MongoDB

MongoDB Mobile allows for building mobile applications to store data in the cloud and on a device. MongoDB Mobile also provides native connection to MongoDB Stitch, giving simple access to the database from devices. Available through MongoDB Stitch, Mobile Sync (beta) allows for changing data offline from mobile devices and automatically synchronizes changes between data held locally and database. Stitch functions and mobile synchronization are available only with the hosted Atlas offering.

DataStax Enterprise (Cassandra)

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

Summary

Unlike DataStax Enterprise (Cassandra), Couchbase and MongoDB provide solutions for supporting mobile applications. Couchbase, Inc. offers Couchbase Lite, which is a full-featured, embedded NoSQL database with built-in security, query, full-text search, and synchronization capabilities that can be used in the offline and online modes. MongoDB introduces solutions for mobile devices support only with the Atlas offering. The functionality to synchronize data changes between devices and MongoDB Atlas is currently available only in a beta version.

Table 3.3.6 Mobile device support in the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	5	0

3.3.7 Logging and statistics

Logging and statistics refer to the capabilities that log performance and errors, as well as 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 CLI utilities (*cbcollect_info*) or Couchbase Server's REST API.

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

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. With the collection stats method, it is possible to get statistics about a collection.

DataStax Enterprise (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 at run time 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 OpsCenter's web UI.

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 3.3.7 Logging and statistics in the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	10	10

3.3.8 Documentation

Documentation embraces user guides, white papers, online help, quick reference guides, etc.

Couchbase Server

All Couchbase Server versions are well documented. The official documentation includes a 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 the company's developers actively contributing articles on a multitude of topics. 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.

Couchbase, Inc. provides learning services available worldwide. Being free-of-charge, online trainings are based on real-life use cases and aim at improving skills necessary for building applications.

MongoDB

The comprehensive product documentation is available via the official MongoDB website. It includes all the required guidelines both for developers and administrators. Publicly available information also includes technical and marketing white papers, webinars, datasheets, presentations, events, a blog, 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 database administrators. Both public and private trainings taught by MongoDB engineers are provided on demand.

DataStax Enterprise (Cassandra)

The documentation for the latest version of Cassandra is available on its official website. It covers major developing and operational topics required to use 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.

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

Summary

All three databases provide extensive documentation and materials for understanding all the concepts behind each database.

Table 3.3.8 Evaluating documentation of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	10	10

3.3.9 Integration

Integration refers to the ease of linking 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 and data drivers that support analytic/business intelligence platforms.

MongoDB

MongoDB has a connector for the Hadoop ecosystem that provides a possibility 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 data using the existing relational BI tools. Numerous open-source projects provide their own integrations with MongoDB.

DataStax Enterprise (Cassandra)

DataStax Enterprise (Cassandra) provides integration with Apache Software Foundation projects, such as Hadoop, Spark, Kafka, and Solr. In addition, Cassandra supports Graph and its own distributed file system—DataStax Enterprise file system (DSEFS). Cassandra has its own implementation of Spark integration for real-time and batch analytics capabilities, as well as 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, Inc. and third-party vendors.

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 communities. All the solutions support popular programming languages through dedicated drivers.

Table 3.3.9 Integration capabilities of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	10	10

3.3.10 Usability

Usability refers to the ease of efficient usage.

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.

MongoDB

MongoDB provides various tools to automate deployments—Atlas, as well as Ops and Cloud managers. The tools differ in features and compatibilities with other MongoDB solutions and deployment options. To choose a tool for a deployment, developers are forced to make some effort to compare the tools and analyze its features and compatibility with other solutions provided by MongoDB.

Atlas, as well as Ops and Cloud managers, significantly simplify administrative tasks, though, it lacks a fully integrated architecture, still requiring careful planning and strong knowledge of the database architecture and its components.

DataStax Enterprise (Cassandra)

DSE provides a proper set of tools for the database operators and programmers. In addition to monitoring and maintenance capabilities, OpsCenter enables you to automatically set up the entire cluster, schedule backups and recover from them, run anti-entropy repairs, etc. Being 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.

Summary

All the three products provide UI tools for cluster and data management. MongoDB is more complicated in its use due to the relatively sophisticated and non-integrated architecture. MongoDB offers various tools for cluster management that differ in the provided functionality and have their own pros and cons, making it difficult to choose between them.

Table 3.3.10 Usability of the NoSQL data stores on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	8	10

3.3.11 Support

Support refers to the active community around the databases, as well as effectiveness and speed of getting help regarding any problem with product from the official support team.

Couchbase Server

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

Couchbase, Inc. also offers community forums. These forums can be used for asking questions, learning, and sharing knowledge about Couchbase products with other developers, administrators, and in-house experts.

MongoDB

Comprehensive MongoDB support programs are provided both by MongoDB, Inc. and various third-party companies. Apart from commercial support, there is a reasonably large user community on popular platforms, such as StackOverflow, ServerFault, Community Support Forum, MongoDB Slack Community, IRC Chat and Support, and DBA Stack Exchange where you can get help and exchange your ideas with others.

DataStax Enterprise (Cassandra)

DataStax, Inc. provides several kinds of subscriptions. There are DataStax distributions of Apache Cassandra, DataStax Basic, DataStax Enterprise, and DataStax Management Cloud. Each subscription includes different kind of features.

DataStax, Inc. supports DataStax Help Center. Through this center, you may ask a question and start/view a discussion about DataStax products.

Summary

All the three solutions under comparison have commercial and free-based support, as well as numerous responsive communities.

Table 3.3.11 Enterprise and community support on a scale of 1–10

Couchbase Server	MongoDB	DSE (Cassandra)
10	10	10

4. Evaluation results

The table below summarizes the points scored by DataStax Enterprise (Cassandra) v6.7, Couchbase Server (v6.0), and MongoDB (v4.0) for each criterion on a scale of 1–10. Here, we assumed 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.

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 for mobile/offline-first web applications. Cassandra works well for write-intensive workloads on a large scale for structured data (including non-relational data).

Couchbase's multi-dimensional scaling feature targets specific customer scenarios that need to quickly scale up/out different workload types (either key-value, indexing, or full-text search). The concept seems attractive for deployments at a scaling to hundreds of cluster nodes. N1QL extends SQL, providing extra functions for querying JSON documents. XDCR allows for simple and reliable design of 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 workloads 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 the exceptional role of configuration servers. All the mentioned points significantly complicate the database's deployment design and maintenance. 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 materials to learn from.

Table 4.1 The comparative scores of Couchbase Server vs. MongoDB vs. DataStax Enterprise (Cassandra)

Criteria		Couchbase Server		MongoDB		DSE (Cassandra)	
Definition	**Weight**	**Basic Score**	**Weighted score**	**Basic Score**	**Weighted score**	**Basic Score**	**Weighted score**
Architecture							
Topology		10		7		8	
Scalability		10		7		10	
Replication		10		8		10	
Consistency		9		9		9	
Availability		8		6		10	
Server and network fault tolerance		9		7		10	
Administration							
Configuration management		10		10		10	
Backup		9		9		9	
Disaster recovery		10		9		10	
Maintenance		10		8		9	
Recovery		10		9		10	
Monitoring		10		10		10	
Security		9		10		9	
Development							
Data structure and format		10		10		8	
A query language		10		7		6	
Full-text search		9		7		9	
Analytics		9		7		9	
Events and triggers		9		8		5	
Mobile device support		10		5		0	
Logging and statistics		10		10		10	
Documentation		10		10		10	
Integration		10		10		10	
Usability		10		8		10	
Support		10		10		10	
Total Score		**231**		**206**		**214**	

5. About the Authors

Artsiom Yudovin is Data Engineer at Altoros, having a solid software development background. He is focused on maintaining, designing, customizing, upgrading, and implementing complex software architectures, including data-intensive and distributed system. Furthermore, Artsiom dedicates much of his spare time to this activity and now he is one of the contributors to famous open-source projects.

Yauheniya Novikova is Senior Java Engineer at Altoros, possessing extensive experience in designing and implementing high-load enterprise applications. She has a strong background in both building software systems from scratch and customizing existing solutions, as well as in designing, analyzing, and testing them. Yauheniya demonstrates an ability to solve complicated issues using the latest big data technologies.

The research paper is edited and published by Sophia Turol, Carlo Gutierrez, and Alex Khizhniak.

Altoros is a 300+ people strong consultancy that helps Global 2000 organizations with a methodology, training, technology building blocks, and end-to-end solution development. The company turns cloud-native app development, customer analytics, blockchain, and AI into products with a sustainable competitive advantage. Assisting enterprises on their way to digital transformation, Altoros stands behind some of the world's largest Cloud Foundry and NoSQL deployments. For more, please visit www.altoros.com.

To download more of NoSQL research papers and articles:

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