Cassandra vs MongoDB Comparison

If you're having trouble scaling with Cassandra, test the easiest way to run a database in the cloud and explore MongoDB Atlas free today.

Get started free

When NoSQL databases first launched in the early 2000s, they were immediately adopted by some of today's software giants. These organizations understood the growing power of data and the implications that power would have on modern applications. However, in those days, NoSQL databases were not considered general-purpose databases, like how we describe MongoDB today. Each one has been developed to answer a demand for some specific workload requirement.

One of these early NoSQL databases is Cassandra, a distributed database based on a hybrid between a tabular store and a key-value store. MongoDB’s distributed document data model is a proven alternative to Cassandra, as it can be adapted to serve many different use cases. In this article, we will discuss the differences between Cassandra and MongoDB.

Since Cassandra has different distributions, we are going to focus specifically on Apache Cassandra in this article.

Table of Contents

What is MongoDB?

MongoDB is a document database built for general-purpose usage. It stores data in an optimized JSON format called BSON (Binary JSON). This allows the MongoDB document model to benefit from the flexibility of JSON documents stored in logical groups called collections. The document model supports many different data types, including strings, dates, numbers, arrays, decimals, nested objects, geo data, and binary data.

MongoDB uses replication and data partitioning called sharding to distribute data for high availability and scalability purposes, making it a highly consistent and fault tolerant database. MongoDB supports multiple deployment methods, from running it for free on a variety of platforms and servers starting from your local machine to a fully blown deployment in the cloud of your choice using MongoDB Atlas. Additionally, MongoDB Enterprise Advanced and MongoDB Atlas offer enterprise-grade security features like authentication, authorization, and LDAP support as well as end-to-end encryption.

MongoDB's unified query API and powerful aggregations, in conjunction with the flexibility of the document model, has made it the most popular general-purpose document database on the market.

What is Cassandra?

Cassandra was initially started by Facebook in 2008 and was intended to power their inbox search capabilities. It is considered to be a distributed, wide column store running as clusters of nodes called "rings." Each node in a Cassandra ring stores some ranges of the data and replicates other ranges as a scaling and fault tolerance mechanism. Cassandra is an eventually consistent database with a limited tunable consistency, but in general, it favors availability over consistency.

Over the years, Cassandra was open-sourced and is now part of the Apache Foundation, while there are additional commercially available versions.

The data is stored in tabular-like tables, with a unique identifier (or set of combined unique identifiers) acting as a key to a stored row, operating mostly as a key-value store with the ability to add a large number of columns to the key. The amount of columns can vary from row to row and therefore, Cassandra has to manage some metadata attributes for each row.

Cassandra's query language is called CQL, and it has its share of similarities to SQL. However, Cassandra does not support joins or subqueries and therefore requires a developer to denormalize the data or duplicate data for efficient access.

Cassandra vs MongoDB: Comparing Key Differences

Let's highlight the main differences between Cassandra and MongoDB in the table below.

	Cassandra
Data Model	Cassandra stores data in a wide column tabular store. Each row contains a unique identifier (or set of identifiers) as its primary key and a following set of columns. The primary key serves as a partition hash key as data is distributed across the cluster.
Indexing	Cassandra offers standard built-in indexing as well as basic secondary indexes to index additional columns to allow queries filtering. Maintaining a large secondary index can potentially cause scalability issues as the cluster grows.
Query Language	Cassandra uses a proprietary language called CQL, which is similar to SQL. Applications can use different drivers provided mostly by third parties and a shell.
Transactions	Cassandra does not support transactions.
Concurrency	Cassandra isolates on a row-level. This means that a specific row for a specific partition can be operated simultaneously by one client. Cassandra also offers tunable consistency for writing, when a client changes its consistency level (one, quorum, all).
	Cassandra replicates its keyspace across the cluster nodes based on the keyspace replication factor.

Cassandra

High Availability and Scalability	Since it's a multi-master ring, each node is holding a range of the partition key per table, while also hosting parts of other nodes' replicas. Each node coordinates reads to the correct node to retrieve or operate the data while it also needs to repair data that got out of consistency across the nodes. The only way to replace the default hash partitioning is by changing the partitioner component for the entire cluster.
Security	Apache Cassandra supports the following basic security methods: • TLS/SSL support for client connections • User authentication • User authorizations with roles
Mobile Support	There is no specific Apache Cassandra version or tools for mobile development. There is also no specific mobile oriented driver or SDK.
Cloud Offerings	Provided via third-party services on different clouds and platforms. Compatibility tests to the Apache Cassandra version need to be verified with each vendor.
Documentation & University	Apache Cassandra offers documentation on their main website, including a dedicated community. There are no official university or online courses on the Cassandra website.
Native Data Visualization Tooling	There are no native data visualization tools provided by the Cassandra project.

MongoDB

Data Model	MongoDB stores documents in an optimized BSON format. Documents are grouped in databases and collections and returned as JSON documents with support for a large number of data types, including: strings, numbers, geo data, dates, arrays, decimal, nested objects, and binary data. Read more on data modeling with MongoDB here.
Indexing	MongoDB supports many index types for various use cases. Secondary indexes on any field are available and supported in different types: Compound, Text, Geo, TTL, Partial, Wildcard and Compound Wildcard indexes.
Query Language	MongoDB has a rich query language called MQL. It supports a wide variety of modern native drivers as well as a shell. The data can be edited, deleted, inserted, and queried in many shapes and forms. The queries can use complex operators. Additionally, the aggregation framework allows you to aggregate data with many stages.
Transactions	MongoDB supports fully ACID compliant transactions.
	MongoDB allows multiple database users to concurrently access the same data by managing a well defined concurrency control.

	MongoDB
Concurrency	MongoDB uses document-level locking, so writes to a single document occur either in full or not at all, and clients always see consistent data. Together with those mechanisms, MongoDB supports different read and write concerns for distributed clusters and retryable reads and writes.
High Availability and Scalability	MongoDB was built from the ground up to support distribution of data using replication and sharding mechanisms. Replica sets host an identical copy of the data and elect a primary which receives all the writes, while other nodes are secondaries replicating all the data. Sharding allows you to easily scale your collections across multiple replica sets. With geo-zone sharding, you can also easily manage data sovereignty requirements. The ability to define specific shard keys and reshard collections with zero downtime when a shard key is no longer optimal gives your application a huge advantage when managing massively distributed datasets at scale. The shard key advisor commands will help you refine your shard keys.
	MongoDB supports enterprise-grade security mechanisms to secure your MongoDB deployments. Most of them are on by default in MongoDB Atlas cloud offering:

	MongoDB
Security	• Authentication and authorization using built-in SCRAM or certificates • TLS/SSL, x509, Queryable Encryption and Client Side Field Level Encryption • Server Side storage engine encryption • LDAP and Kerberos integrations Additionally, MongoDB cloud offerings have strong security compliance certifications. Read more on our trust center.
Mobile Support	Atlas for the Edge streamlines data management between the database and edge devices. Within this solution, you will find two key components for mobile development: Atlas Device SDKs and Atlas Device Sync Atlas Device SDKs, available for most popular languages, frameworks and platforms, offer a lightweight reactive on-device object-store for mobile/edge/IoT devices. Atlas Device Sync offers offline-first syncronization between MongoDB Atlas clusters and the on-device database with automatic conflict resolution and strong eventual consistency. Changesets may arrive any time that connectivity allows.
	MongoDB Atlas, the *database-as-a-service* platform, offers clusters in all three major cloud providers, starting from a free tier to a fully blown production cross-region and cross-cloud cluster.

	MongoDB
Cloud Offerings	Together with Atlas, you get the advantages of using Atlas App Services application services, Charts, and Data Federation for querying your data lake alongside Atlas clusters, as well as Atlas Search for optimized full text search.
Documentation & University	Detailed documentation with examples and full tutorials including a full community and developer hub websites. Online university with some free courses available at https://learn.mongodb.com.
Native Data Visualization Tooling	MongoDB Charts provides a quick, simple, and powerful way to perform data visualization with MongoDB Atlas data. Additionally, MongoDB Compass provides a GUI client for MongoDB. MongoDB also provides a connector to integrate with popular third-party business intelligence tools.

Cassandra vs MongoDB: Supported Languages

Both Cassandra and MongoDB recommend using language native drivers to interact with their databases. MongoDB offers a set of officially supported drivers for almost every modern language, including Atlas Device SDKs for the edge on-device database. It also offers an extension of community drivers and ODMs (Object to Document Mapping).

Cassandra offers different flavors of drivers with each language, as they are mostly supported by third-party vendors.

MongoDB publishes an up-to-date compatibility matrix to verify that clients use an optimal driver version for each associated server version. In addition, MongoDB 5.0+ supports a Stable API, which allows driver and server version to be decoupled as long as the API version is aligned. This provides better and safer upgrade guarantees with no breaking changes.

Cassandra documentation does not offer a clear compatibility picture and developers are dependent on third-party documentation to understand the version requirements.

Cassandra vs MongoDB: Query Language

MongoDB has a single and robust query API called MQL (MongoDB Query API). It uses different CRUD methods with JSON object inputs to describe queries, write operations, and aggregations.

Cassandra uses a query language called CQL, which has similarities to SQL, as it uses similar keywords like “SELECT,” “INSERT,” “UPDATE,” etc., to interact with Cassandra tables.

Let's compare similar database commands between Cassandra and MongoDB.

Create a row/document

In Cassandra:

```
INSERT INTO planets (id, name)
   VALUES ("uhgyrt-shhwyey-shhday-ueyr", 'Earth");
```

In MongoDB:

```
db.planets.insertOne({"name" : "Earth"});
```

Comparison Notes: With both methods, we submit the record to be stored in “planets” (Table/Collection).

Query a document by Id

In Cassandra:

```
SELECT * FROM companies WHERE id = "8843faaf0b831d364278331bc3001bd8";
```

id	name
"8843faaf0b831d364278331bc3001bd8"	"Example, Inc."

In MongoDB:

```
db.companies.findOne({"_id" : "8843faaf0b831d364278331bc3001bd8"});

{
"_id"  : "8843faaf0b831d364278331bc3001bd8",
"name" : "Example, Inc."
}
```

Comparison Notes: Retrieving the data in MongoDB is done with a query parameter, whereas in Cassandra, it's done via a SELECT statement.

Update a document

In Cassandra:

```
UPDATE albums
   SET title = "One by One", "year" = 2001
   WHERE id ="6e1295ed6c29495e54cc05947f18c8af";
```

In MongoDB:

```
db.albums.updateOne({"_id" : "6e1295ed6c29495e54cc05947f18c8af"}, {$set : {title : "On
```

Comparison Notes: To update a record in MongoDB, we need to issue an updateOne command and specify the new field values under the $set operator in the update clause.

In Cassandra, we use the UPDATE command. The where clause must include the entire primary key. Otherwise, the update won't work.

Aggregate a group by tag

In Cassandra:

```
SELECT tag, COUNT(1) as count
FROM products
```

```
GROUP BY tag;
```

In MongoDB:

```
db.products.aggregate([{$group : {_id : "$tag" , count: { $count:{ } } }}]);
```

Comparison Notes: Aggregations with MongoDB are a query language API receiving a pipeline of stages.

Cassandra can use a GROUP BY clause in a SELECT statement. It only accepts primary key columns in defined order as arguments. Otherwise, other methods are required (like custom functions and analytical nodes).

Cassandra vs MongoDB: Data Model

MongoDB's data modeling is done through its document data model. Documents are stored in a Binary JSON format called BSON and are grouped into collections. This allows documents to maintain a flexible and polymorphic structure, as they can adapt to changing requirements in an application's code. This is a departure from a tabular data model, where the schema usually drives how developers must build their application. For example, a soccer gaming application might have the following document:

```
{
  _id: ObjectId("61daf0c600f0af793e477e7b"),
  matchId: ObjectId("61daf0c600f0af998e470e6a"),
  players: [
    {
      id: "61daf0c600f0af793e455e71",
      name: "ItsFootball",
      level: "Advanced"
    },
    {
      id: "61daf0c600f0af793e455e60" ,
      name: "ItsSoccer" ,
      level: "Beginner"
    }
  ],
  startDate: ISODate("2021-01-01T:10:00:00"),
  gamesPlayed: [
    {
      gameId: 1,
      score: { ItsFootball : 1, ItsSoccer : 0 }
  ],
  deviceDetails : {
    name : "Playstation",
    version : "PS5"
  }
}
```

As shown above, a complex state and detailed description can all fit into one logical object. If a data model calls for a stricture structure and finer schema controls, developers can leverage MongoDB's schema validation feature to enforce such rules in the collections which need them.

Cassandra uses a tabular-like structure, where each table resides in a namespace called "keyspace." Keyspaces are similar concepts to databases in MongoDB and provide a grouping level for tables. Tables are structured in a key and wide-column representation, where the primary key defines the key structure. Without a secondary index, data can be either fully scanned or queried via a primary key filter. Columns can vary from row to row, but require the Primary key as part of each row.

A representation of the previous presented document as a Cassandra row could be defined as:

MatchId (PrimaryKey)

Player1_id

"61daf0c470e6a"	"61daf0c455e71"

Player1_name

"61daf0c470e6a"	"ItsFootball"

Player1_level

"61daf0c470e6a"	"Advanced"

Player2_name

"61daf0c470e6a"	“ItsSoccer”

deviceDetails_name

"61daf0c470e6a"	“Playstation”

... etc

"61daf0c470e6a"	... etc

The data represented in this single MongoDB document must be distributed across roughly 17 columns in a Cassandra row.

Cassandra vs MongoDB: Secondary Indexes

Secondary indexes are a vital consideration for any application’s queries. These are vital for enabling an application to performantly fetch data that is not tied to a specific unique identifier or a subset of a primary key.

MongoDB allows developers to build secondary indexes as part of basic collection administration on any field, including objects, arrays, geographic data, and even in conjunction with Wildcard and Compound Wildcard indexes. Additionally, the MongoDB Atlas developer data platform allows users to build secondary full text search indexes. Secondary indexes require no administration once created.

Cassandra has the ability to create secondary indexes on other columns than the defined primary key. However, Cassandra will not allow filtering on other columns without a secondary index, while in MongoDB, the query language can filter on non-indexed fields as well. This dramatically impacts query flexibility as developers must predict all columns used for filtering up-front or create indexes on the fly, risking system stability.

Additionally, secondary indexes in Cassandra are not part of the partition key by definition. This means Cassandra must broadcast a query to all nodes and wait for all to report an answer before it is considered complete. These limitations will lead to Cassandra being more difficult to manage at scale than MongoDB, and especially MongoDB Atlas.

Cassandra vs MongoDB: Availability

MongoDB uses replication and replica sets to assure cluster availability. We recommend that each replica set consists of an odd number of nodes (three as a production minimum), allowing it to be fault tolerant to server failures. Whenever a primary becomes unavailable, an election automatically takes place to allow another replica set member to take over as the new primary. Having a single point that controls and performs all writes allows MongoDB to have tunable consistency levels, varying from strong to no conflict resolution requirements through using read and write concerns. MongoDB also allows reading from secondary replica set members with no block to replication for use cases when performance is valued over consistency.

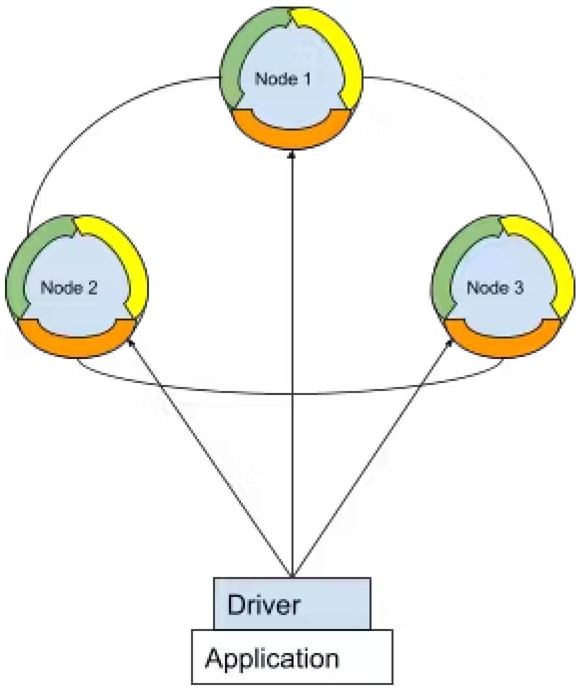
Cassandra uses a replication factor set on a keyspace level to distribute replicas of the keyspace across the different nodes in a ring. The specific factor can be configured by the user. Distributing the replicas in a production environment requires additional architecture considerations as the cluster needs to identify “availability racks” and associate replication to different racks to avoid outages. Since each Cassandra node acts as a partition for one set of data and replicates another, there are consistency and coordination mechanisms that need to govern data distribution among nodes. Those mechanisms, like read repair, can cause performance issues and blocking write periods for replication.

Cassandra vs MongoDB: Scalability

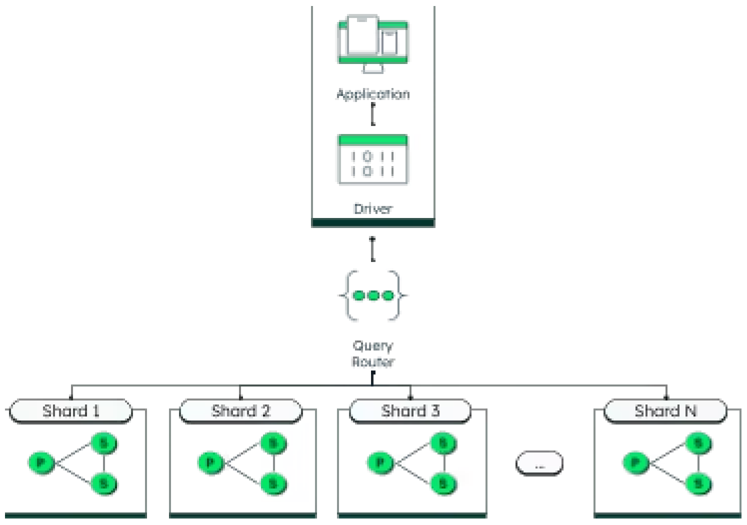
MongoDB was designed to support horizontal scaling and elasticity through sharding. A sharded cluster consists of many replica sets, each holding a piece of the collection. The data distribution is decided by the sharding key and sharding configuration. The developer has the flexibility to choose the shard key and the type of sharding they wish to use (range, hash, etc.). Additionally, via zone sharding, data can be associated to a specific group of shards in accordance with the even distribution concept. MongoDB 5.0 introduced live resharding, which allows you to change shard keys with no downtime, allowing data distribution to evolve along with an application’s evolving needs. MongoDB 7.0 introduced shard key advisor commands, which generate metrics that will help you refine your shard keys.

Cassandra uses the concept of a partition key, which is the table primary key, to evenly distribute data across the cluster. Essentially, Cassandra’s partitioning functionality only supports hash sharding, with no ability to change the shard key over time without undergoing a time-consuming and costly process of manually re-sharding.

Due to its limitation of only offering hash sharding, Cassandra cannot offer cross partition transactions, joins, or data referencing keys.



Cassandra Cluster Data Clustering Diagram



MongoDB Sharding High Level Diagram

Cassandra vs MongoDB: Aggregation Framework

MongoDB offers a robust and rich aggregation framework, to reshape, calculate, and output your data to the client or other collection. The aggregation framework serves as a pipeline of stages where each stage gets an input stream of the previous stage's documents, manipulating them and passing to the next stage. You can find operations like grouping, counting, restructuring objects and arrays, and many more. MongoDB Atlas also offers a built-in aggregation builder as part of its data explorer, support for Atlas Search's $search stage, and aggregations on Atlas Data Federation's data lakes.

Cassandra offers basic aggregations from version 3.0+ as part of its CQL SELECT statement, such as GROUP BY and COUNT. However, more sophisticated aggregations will need to be handled by the development team and built into an application's logic. It is also common for Cassandra users to have to rely on third-party utilities, such as Hadoop, in order to effectively analyze and aggregate large amounts of data.

Cassandra vs MongoDB: Read Performance

MongoDB is optimized for both reads and writes. If a user ensures that the correct indexes are in place for their application's common queries, and especially if the indexes fit in memory, they can expect high read performance which is capable of supporting most modern applications. MongoDB's performance can also be tuned for specific workloads through proper document schema design and cluster topology planning.

High read performance also does not need to come at the cost of strong consistency. Users can tune their cluster's workloads to read from the primary node, which eliminates the need for queries to be dependent on data coordination between multiple nodes. If the primary fails, the replica set will quickly elect a new primary from one of the secondaries, and will allow reads to be issued before the new primary is fully operational for writes.

Cassandra's architecture is optimized for writes, rather than complex read patterns. The database uses a multi-master model where the read of a primary key combination (partition-key) is coordinated across nodes and performed against the relevant node. This process adds overhead to the read operations, depending on which node receives what query range. Additionally, secondary index reads could incur higher latency depending on the number of nodes in the cluster. This is because "non-partition" queries are distributed across the total number of nodes, and will end only once every node provides a response.

Cassandra vs MongoDB: ACID Transactions

MongoDB introduced multi-document ACID transactions from version 4.0 (sharded clusters from 4.2). This support allows applications which require fully ACID cross document transactions to perform all or nothing operations. Transactional databases fit a specific use case requirement, and we still encourage developers to consider alternate schema design prior to using them, in order to avoid adding unnecessary complexity to their application. Transaction support is one of the main differentiators between MongoDB and Cassandra.

Cassandra does not support multi-row ACID transactions, and allows only isolation and durability to be tuned based on a single row operation. Cassandra's consistency model is, by default, in preference of availability over consistency.

Cassandra vs MongoDB: Use Cases

MongoDB is a general purpose document database. Therefore, it has a rich set of features and design patterns to cover almost all of today's modern applications. Many common modernizations to legacy SQL applications use MongoDB, as its flexible schema can be customized to transform existing data into the document model. Developers can also port over existing functionality from their SQL application, such as ACID transactions, secondary indexes on any field, and rich aggregation and query capabilities. In addition, it also supports horizontal scalability for big data as a core feature, as well as high availability and data segregation for meeting Service Level Agreements for latency and compliance needs.

Cassandra is mostly used for key-value columnar use cases. It is more suitable for very predictable reading and writing patterns, especially in write-heavy workloads–for example, a logging or tracking system where there are no or a very small number of in-place updates and not many secondary indexes in place.

With the release of Time Series Collections in MongoDB 5.0+, it is now just as easy and efficient to ingest, store, and query time series data as it is to work with any operational data in standard collections.

Conclusion

Apache Cassandra is a widely adopted wide-column store designed for specific use cases where the writes are done by a single primary key and are the vast majority of workloads. Scaling in Cassandra is only applicable to fairly niche workloads.

MongoDB is a general-purpose database that can support multiple use cases with its flexible document model, rich aggregation language, and robust features such as sharding and ACID compliant transactions. Therefore, it can cover the vast majority of Cassandra's most popular use cases, and much more.

Your real question should be what limits you to not use MongoDB for your next application. Since there is no easier way to run a database in the cloud than MongoDB Atlas, you should get started today.

FAQ

Are MongoDB and Cassandra the same?

No. MongoDB is a general-purpose document database while Cassandra is a wide-column NoSQL store. Many use cases can be covered by each database, but their differing architectures will drastically impact application design, efficiency, and cost to maintain.

Why use MongoDB over Cassandra?

MongoDB offers a larger superset of features and abilities than Cassandra. With the correct schema and design pattern, MongoDB can cover the vast majority of use cases covered by Cassandra. Additionally, the MongoDB Atlas platform is a unique top class developer data platform offered by no other company or technology.

Which is better: MongoDB vs Cassandra?

It depends on the use case. However, there are more use cases where MongoDB will help you improve developer productivity as well as go-to-market times over developing with Cassandra.

How is Cassandra different from MongoDB?

Cassandra uses a cluster of nodes hosting wide-column tables queried by the CQL language. On the contrary, MongoDB stores its data in JSON documents, stored server-side in a binary JSON format called BSON. It uses a single query API called MQL to interact with the data within its databases and collections. Both MongoDB and Cassandra have built-in scalability features and high availability guarantees. However, the semantics and consistency guarantees are different.

Is MongoDB good for reading or writing?

Both. MongoDB is optimized for read and write workloads, with the ability to tune your consistency and read preference. It uses secondary indexes on any field to fit your query patterns and offer differentiating features for workload isolations and data governance to fit your needs.