Couchbase

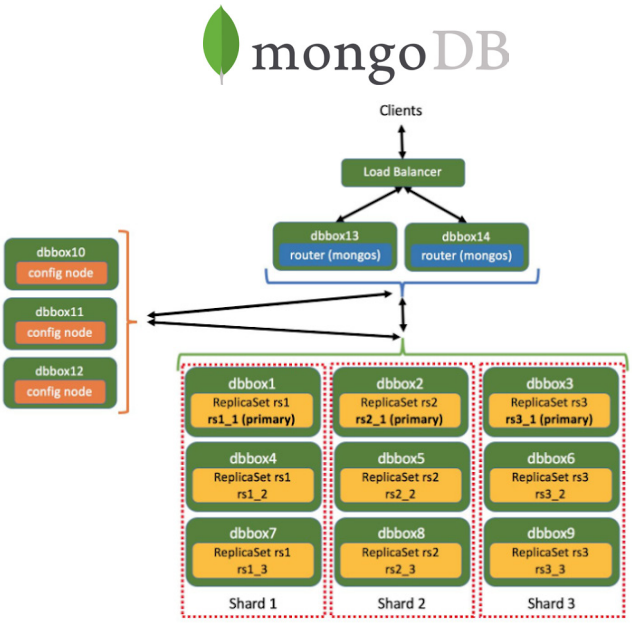
Couchbase vs. MongoDB™ for Scale-Out and High Availability

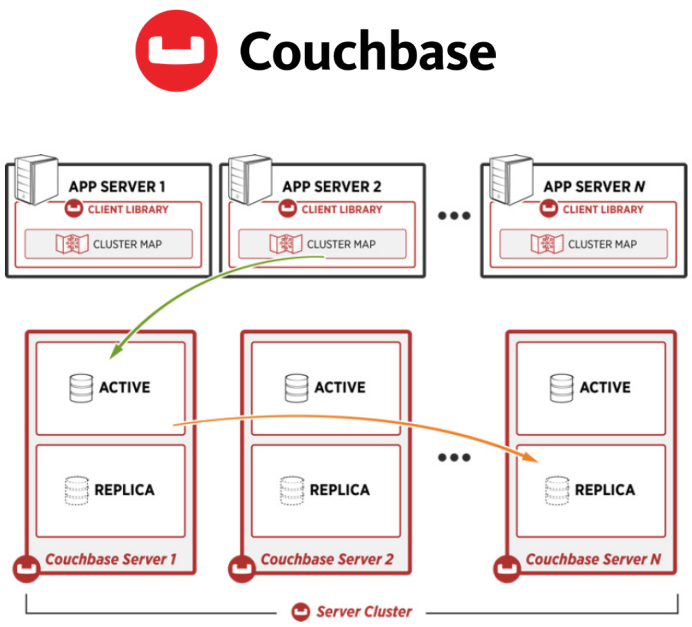
Introduction

Couchbase is a distributed NoSQL document-oriented database with a core architecture that supports a flexible data model, easy scalability, consistent high-performance, always-on 24x365 characteristics, and advanced security. It has been adopted across multiple industries and in the largest enterprises for their most business-critical applications. Many of those customers, including: Carrefour, DirecTV and Equifax, have gone through rigorous evaluations of Couchbase alongside MongoDB, and have chosen Couchbase based on a strong set of differentiated capabilities and architectural advantages, including:

- **Scale-Out and High Availability**
- **Disaster Recovery**
- **SQL/SQL++ Query for JSON**
- **Hybrid Operational and Analytical Processing**
- **Full-Text Search**
- **Server-Side Eventing and Functions**
- **Embedded Mobile Database**

In this paper, we will focus on how MongoDB compares to Couchbase when it comes to enabling enterprise applications to scale easily, efficiently and reliably, scale single services instead of the entire database, and avoid downtime and maintain high availability 24x7 with replication, automatic failover, and online operations.





Scale-out and High Availability: Easy, Flexible and Robust

MongoDB is a hierarchical architecture that forces a complex set up requiring a variety of nodes (config servers, routers, replica sets) in order to scale-out as application workload grows. On the other hand, Couchbase is a true peer-to-peer distributed system with a simple yet flexible architecture to allow for an application's evolving needs.

- Couchbase has automatic sharding versus MongoDB's manual (and laborious) sharding where a shard key has to be chosen per collection.
- Couchbase is flexible and adapts to new workloads by allowing global secondary indexes that are partitioned differently than data. MongoDB is rigid and optimized only for a single access pattern - all other access patterns result in scatter-gather across all nodes.
- Couchbase is scale-agnostic and all features work at all levels of scale. MongoDB on the other hand is biased towards a single-shard set-up and many critical features such as joins and grouping do not work on multi-shard setups.
- Couchbase can scale-out by simply adding a single node. MongoDB on the other hand requires addition of at least 3 nodes to add another shard to the cluster resulting in a very high TCO.
- Couchbase failover is very fast (6-7s) and robust as it is based on multiple signals of liveness. MongoDB failover is much slower (13-14s) and fragile as it is based only on heartbeats between replica-set nodes.

Easy Scale-out and High Availability with Couchbase

MongoDB's origins in a single replica set architecture result in many complexities and inefficiencies when a truly scale-out and highly available database is required for mission critical applications.

- Sharding in MongoDB is either manual or "shard-key" based putting the burden on the application and operations teams to know their data. It also locks the cluster into one access pattern. E.g. for Flights documents containing source and destination airports, if source airport is chosen as shard key, then queries on source airport will be fast, but queries on destination airport will be slow.
- Range sharding can create data skew making the key selection an even more onerous task. Hash sharding in MongoDB on the other hand results in more broadcast operations.
- Mongo collections may be sharded differently, therefore it is not easy to just add or remove capacity by adding or removing nodes. The impact of adding or removing a node is not clear as it depends on how collections are sharded. See the MongoDB documentation on sharding to appreciate how complex it is to design the shard keys and manage the shards, and deal with the shard restrictions.
- MongoDB requires setting up routers, config servers and replica sets in a sharded setup. Furthermore, the network connectivity between the Mongo router, config servers, and each shard influences overall performance as it involves multiple hops.
- Replication set-up for High Availability is complex and wasteful. MongoDB requires a replica-set to be created for each shard requiring a minimum of 3 nodes for production setups. The secondary node servers are idle with low hardware utilization.
- MongoDB does not have built-in Rack Awareness. It must be configured manually by selecting secondaries of each replica set on different racks in a very laborious way.

Couchbase Server, on the other hand, was engineered to scale out easily. Couchbase Server does not require manual configuration of sharding. It is automatic. It is so simple that partitioning of a bucket is described in a single page. Contrast that to the multiple sections MongoDB needs to describe sharding.

Couchbase High Availability setup is so simple that it almost requires no setup:

- There are no replica-sets to be created.
- Couchbase automatically distributes all shards (vbuckets) and their secondaries uniformly across all

nodes.

- Each server node has an equal number of active vbuckets and replica vbuckets.
- Built in Rack Awareness with Server Groups is as simple as assigning a server group to each server and rebalancing.
- There are no idle nodes in Couchbase.

When new capacity is needed, each service can simply rebalance its workload to adapt to the new capacity. Adding or removing a node to add or reduce capacity can be easily automated (as in Couchbase Operator for Kubernetes) without manual intervention.

MongoDB is Rigid and Not Adaptive to Changing Workloads

MongoDB shard key cannot be changed once the collection is created. Choice of shard key is an irreversible critical decision that can make or break your cluster as stated in their documentation:

> *The choice of shard key affects the performance, efficiency, and scalability of a sharded cluster. A cluster with the best possible hardware and infrastructure can be bottlenecked by the choice of shard key. The choice of shard key and its backing index can also affect the sharding strategy that your cluster can use.*

MongoDB forces data and its indexes to have the same partition key. As long as the partition key is provided in a query or primary key lookup, applications can streamline latency. If new workloads are added that require filtering by non-partition keys, MongoDB has to fall back to scatter-gather. Scatter-gather can introduce higher processing overhead and longer latency.

> MongoDB documentation *states the limitation of scatter-gather as follows: "If a query does not include the shard key, the mongos must direct the query to all shards in the cluster. These scatter gather queries can be inefficient. On larger clusters, scatter gather queries are unfeasible for routine operations."*

Scatter-gather is an anti-pattern for scale-out architecture whenever the performance degradation of operational queries is associated with the growth of cluster size. When business grows and requires larger cluster, it could proportionally slow down the application.

Also, since MongoDB does all query processing locally on data nodes, as operational query workload grows, it requires scaling of the data cluster (resulting in movement of data chunks) even though there is no change to data size. MongoDB provides no workload isolation or independent scaling for operational query workloads.

Couchbase is Architected for Change

Couchbase does not require any irreversible decisions such as selection of shard keys. Additionally, the Global Secondary Index service provides partition independence: data and its indexes can have different partition keys. Data (documents) is automatically and transparently partitioned by primary key to optimize for primary key lookup. Indexes can have their own partition key (or none at all), so each can be partitioned independently to match the specific query. As new requirements arise, the application will also be able to create new indexes with their own partition keys, without affecting the performance of existing queries. This gives applications the power to reduce latency at scale (by eliminating scatter-gather), along with the flexibility to optimize for different access patterns side-by-side.

With data and index separation, application can add as many indexes as needed without affecting write latency. This de-couples latency optimization between write traffic and read traffic (query). Applications do not have to worry that adding new indexes will slow down write traffic, yet can still maintain consistency on read when required. This level of adaptability works particularly well for microservices as the database can continue to evolve without worrying about degradation of write performance for critical business transactions (e.g. submit order).

Another advantage of the Couchbase architecture is that Global Secondary Indexes can be deployed on a different set of nodes than the data nodes. This allows indexing nodes to be scaled independently as query workload

grows. Couchbase refers to this as Multi-Dimensional Scaling (MDS).
MDS also enables selection of hardware to match the workload characteristics. For example, if indexing lookup benefits from more memory, then a different type of machines can be chosen for indexes versus the data nodes. With cloud computing, it is easy to match the resources with a specific workload. With Couchbase, applications can benefit from true workload isolation within the database where each service can run on different instance type for specific performance and need. This gives full power to the application to adjust its workload performance according to evolving needs.

Most importantly, however, is that this level of isolation and segregation is possible but not required. A single node of Couchbase on your laptop exposes the exact same APIs and behavior as a 3-node test cluster of VMs in your datacenter and behaves exactly the same as a 20+ node cluster of cloud instances or containers. No application changes are required to deploy across a multitude of environments. Unlike MongoDB, all features of Couchbase are available and behave the same whether using a single node setup or a multi-node setup.

TCO for a MongoDB Sharded Set-Up is Much Higher

The number of servers required for a sharded MongoDB set up is much higher (at least 3x more than Couchbase) than the number of shards.

- For an HA setup, each shard in MongoDB requires building a replica-set with a minimum of 3 nodes. Not only is replica-set setup complex, it is expensive due to the number of servers required. An Arbiter node can be used instead of a full data copy, but this still requires a full node to be set aside.
- The secondary nodes of each replica-set (at least 2 nodes per replica-set) sit idle and do not handle any write traffic. This results in under utilized hardware.
- Config Servers have to be setup. These are additional server nodes and a minimum of 3 is required since they must be setup as a replica-set for high availability.

Couchbase, on the other hand:

- Has no requirement on minimum nodes nor for that number to be even or odd
- Transparently distributes active and replica shards evenly across any size cluster
- Has no concept of idle/passive nodes, the workload is evenly shared by all members of each service
- Has no dedicated config/metadata nodes

MongoDB is Not Scale-Agnostic

MongoDB functionality is different depending on whether a collection is sharded or not, hence forcing application changes when going from single node to a sharded setup. This is a result of design choices that MongoDB has made with its sharding and indexing architecture.

MongoDB applications face many changes when switching to a sharded environment, some of which are:

- Application initialization and set up code has to change to use sharding explicitly. E.g. sharding has to be enabled, shard key has to be determined, an index has to be created on the shard key, each collection has to be explicitly sharded and the list goes on. MongoDB has a long documentation page on how to convert to a sharded setup.
- There are many operations that are not available in sharded setups e.g. group and geo-search do not work in a sharded setup. Hence, the application has to be re-written to workaround these limitations. The full list of sharded cluster operation limitations is available on MongoDB documentation. These limitations also highlight the fact that MongoDB's roots lie in a single node system that gradually morphed into a sharded system instead of starting out as one.
- Even functionality such as Unique Keys, that they tout as an advantage over Couchbase, does not work in sharded setups.
- MongoDB's answer to JOIN is $lookup which does not even allow joining across sharded collections with

$lookup: the from collection cannot be sharded.

Couchbase Failover is Fast and Robust

MongoDB failover is based on holding elections when the primary node fails. This adds more time to the failover duration compared to Couchbase which does not need to hold any elections during failover.

MongoDB documents that the median time to elect a primary should not typically exceed 12 seconds which includes time to detect unavailability of primary and to hold an election. Since Couchbase does not need to hold any elections, this overhead is eliminated and Couchbase can process the failover as soon as it detects a failure. Couchbase timeout for failure detection can be configured down to 5 seconds (Couchbase Automatic Failover). It takes Couchbase a couple of seconds to process the failover bringing the total failover time to 6-7 seconds. MongoDB does not document the time it takes to actually process the failover after elections, but even with a best case of 1-2 seconds, their overall failover time is 13-14 seconds in contrast to Couchbase 6-7 seconds. Hence, total downtime with Couchbase is much lower and can make the difference between an application achieving 99.999 versus 99.9 availability.

MongoDB failure detection is fragile and based on heartbeats only. This is the reason MongoDB does not strongly recommend lowering electionTimeoutMillis from the default of 10s as it may result in frequent elections (documentation source here):

> *"Lowering the electionTimeoutMillis replication configuration option from the default 10000 (10 seconds) can result in faster detection of primary failure. However, the cluster may call elections more frequently due to factors such as temporary network latency even if the primary is otherwise healthy. This can result in increased rollbacks for w : 1 write operations."*

In contrast, Couchbase failure detection is more robust and is based on several different signals as opposed to just heartbeats between a subset of nodes:

- Complete view of cluster from multiple observers as opposed to just members of replica-set. Couchbase orchestrator compares heartbeats from all nodes before declaring a node as unavailable, hence making it more resilient to temporary network latency.
- Monitoring of replication traffic between data nodes (including periodic no-ops). This serves as an additional signal of liveness (or not) in case heartbeats don't get sent or processed.
- Disk read/write failures - Couchbase monitors read/write errors and if a node has significant rate of failure, it is auto-failed over. . MongoDB cannot detect unavailability due to disk read/write failures and will keep sending operations to that node eventually requiring manual intervention.

Couchbase provides the same Auto-failover mechanism for all services and nodes including Data nodes, Index nodes, and Query nodes. Auto-failover is enabled by default.

No Fault Isolation in MongoDB

MongoDB is a monolithic process that handles data read/write, cluster management, index management and query processing. A failure in any one of these domains causes all the functionality of the node to go down as the single process architecture provides no fault isolation.

Couchbase, in contrast, is built on a foundation of separation of concerns with a Service based architecture and multi-dimensional scaling (MDS). A Cluster Manager runs automatically on every node and is responsible for process startup and monitoring, statistics gathering, REST APIs and the management console. It provides a unified control plane for and view into the cluster. The core database functionalities (called services) of Data storage, Global Secondary Indexing, the Query Engine, Full-Text Search, Analytics, and Eventing are all separate processes that are managed by the Cluster Manager. In Couchbase, failure or degradation of one service/component does not impact the behavior or availability of the node or cluster as a whole.

This fault isolation in Couchbase is key in getting even higher levels of uptime from applications running on Couchbase.

Summary of Couchbase Scale-Out and HA versus MongoDB

	Couchbase	MongoDB
Sharding	Automatic for entire dataset	Manual key selection per collection
Data Skew	None. Data is always uniformly distributed	Range sharding can result in non-uniform distribution
Adding/Removing a shard	Simple and one step. Just add one node and all data stretches	Complex. Need to create 3 node replica set. Each collection scales differently
Rack Awareness	Built in and easy. Just create Server Groups	Not built in. Need to manually allocate replicate set nodes from different racks.
Balanced Set Up	Cluster is always balanced with each node having equal number of active vbuckets.	Not balanced. Secondary nodes do not serve any write traffic (not even read traffic by default).
Performance Impact of Adding and Maintaining Indexes	None. Indexes are maintained asynchronously and can be distributed across separate hardware if appropriate.	High. Indexes slow down writes as their are maintained in the write-path.
Index Sharding	Flexible with Global Secondary Keys	Rigid with indexes forced to be partitioned on same key as shard data.
Adaptability to New Workloads	Yes. Can create secondary indexes partitioned on keys that match filtering requirements of new workload.	No adaptability. New query workloads will have to pay the price of scatter-gather and take a performance hit.
Index Scale-Out	Independently scale indexes of data. Can even use different kind of hardware for index nodes	Index scaling tied coupled with data scaling. Have to add capacity to data cluster to accommodate changes in query workload.
High-Availability TCO	None. No special nodes need to be allocated for replicas/secondaries.	Separate nodes need to allocated for secondaries. Minimum of 2 secondary nodes are needed for each shard.
Cluster Metadata	No special nodes needed - distributed across all data nodes.	Special configuration servers nodes need to be set up - a minimum of 3 nods
Application Transparency to Aharding	Applications are fully agnostic to sharding - no changes whether running on 1 node or multiple nodes	Application needs to change when going from single node to a sharded stup
Feature Availability in Sharded Clusters	All features are fully available in sharded clusters. No restrictions whatsoever.	Many key features are not available in sharded collections: such as Group, $lookup, Geo- search, Unique Index

	Couchbase	MongoDB
Automatic Failover Expediency	Very fast. Median time is 6-7 seconds if timeout is lowered to 5 seconds.	Needs to hold elections, hence much slower. Median time works out to 13-14 seconds(with generous benefit of doubt).
Automatic Failover Robustness	Very robust as multiple signals used to make decision(heartbeats from all cluster nodes, replication traffic between nodes, disk errors)	Fragile as it is only based on heartbeats between replica set members.
Fault Isolation	Service architecture and MDS allow complete fault isolation between Data Service, Cluster Manager, Index and Query	No fault isolation - mongod is monolithic and a failure in any domain causes everything to go down

Conclusion

MongoDB provides a long list of checkbox features, but many fail to work in concert with each other, leading to a database that cannot scale, perform, nor adapt to meet today's enterprise requirements. Ultimately, MongoDB is best suited for flexible data access where low latency, high throughput, multiple access patterns, geographic replication, or offline access are not required.

Couchbase, on the other hand, is routinely used for caching layers, sources of truth, and systems of record across high-scale and high-flexibility use cases, including offline-first mobile applications. By design, Couchbase is accessed and managed through a consistent set of APIs, and scaled, upgraded, and diagnosed as a single unit, making Couchbase a complete database platform that not only addresses the needs of today, but offers the flexibility to adapt to the needs of tomorrow.

Learn more

To learn more, contact your Couchbase sales representative today or visit:

couchbase.com | couchbase.com/downloads

Couchbase's mission is to be the data platform that revolutionizes digital innovation. To make this possible, Couchbase created the world's first Engagement Database to help deliver ever-richer and ever-more-personalized customer and employee experiences. Built with the most powerful NoSQL technology, the Couchbase Data Platform was architected on top of an open source foundation for the massively interactive enterprise. Our geo-distributed Engagement Database provides unmatched developer agility and manageability, as well as unparalleled performance at any scale, from any cloud to the edge.