

MongoDB & Redis: NoSQL Big Data Management

Exploring basic theoretical notions regarding NoSQL, together with MongoDB and Redis as implementations and their storage models



What is NoSQL?

- Informal definition: Not Only SQL
- Foundations laid in 2009 during a meetup in San Francisco
- A new databases paradigm that allows for schema-less, semi-structured data with distributed storage



Why NoSQL?

- Traditional RDBMS struggle with huge workloads that occur when dealing with Big Data
- NoSQL advocates claim that the horizontal scalability that comes with this technology, allow for better data management
- In addition, flexibility suits better agile environments where data structure can change over time



CAP Theorem and NoSQL

- ❖ (C) Consistency, (A) Availability, (P) Partition Tolerance
- ❖ Proposed by Eric Brewer in 2000, it claims that between the above three properties on shared systems, only two can be achieved at once
- ❖ As we will see eventually, both Redis and MongoDB are considered CP systems, even if both now implement additional techniques

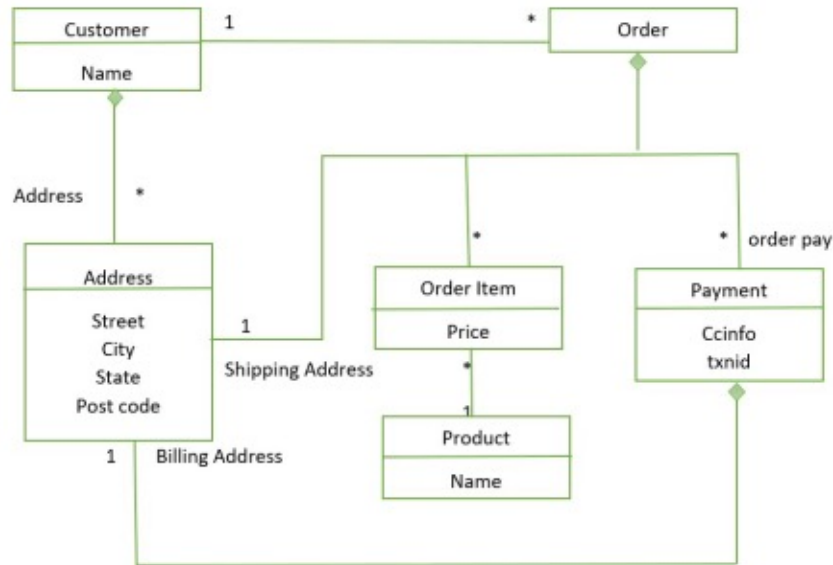


NoSQL Data Models

Type	DBMS Implementation	Storage Model
Key-Value	Redis	Key-Value pairs
Document	MongoDB	JSON-like documents



Aggregates and Schema Flexibility (1)



The foundational units in a NoSQL storage are identified as aggregates.

In the context of NoSQL, it is important to correctly identify aggregates in data when modelling information, so that querying and distribution can be performed efficiently

*Two main aggregates: Order and Customer
A single logical address appears 3 times in the data*



Aggregates and Schema Flexibility (2)

```
_id: ObjectId('67b114361b61cc4d0a2bdcb7')
title: "Broadway Love"
year: 1918
cast: Array (3)
genres: Array (2)
href: "Broadway_Love"
extract: "Broadway Love is a 1918 American silent romance film directed by Ida M..."
thumbnail: "https://upload.wikimedia.org/wikipedia/commons/thumb/0/04/Broadway_Lov..."
thumbnail_width: 320
thumbnail_height: 430
```

In MongoDB, an aggregate is represented by a single document

```
127.0.0.1:6379> HGETALL movie:65041
1) "total_votes"
2) "24.0"
3) "filmtv_id"
4) "65041"
5) "tension"
6) "2.0"
7) "genre"
8) "Action"
9) "effort"
10) "1.0"
11) "public_vote"
12) "4.0"
13) "actors"
14) "Taylor Lautner, Marie Avgeropoulos, Rafi Gavron, Adam Rayner, Luciano Acuna Jr., Josh Yadon, Sam Medina, Jos\xc3\xa1 B\xc3\xa1ez, Woon Yung Park, Johnny M. Wu"
15) "notes"
16) "The plot and psychological development of the characters proceed by stereotypes but it is the tribute that must be paid to a genre that is also reflected in the breakbeat and progressive soundtrack and in the wild rhythm of Peter Amundson's editing. Respecting the packaging, Ben mayor focuses everything on the surprise effect of a well-written neuralgic twist and on the undeniable charisma of Lautner's body. Although some protagonists get lost in the street and gangster moments make you smile, Tracers is among the best products of the recent acrobatic action. We wish Lautner to become an icon."
17) "duration"
18) "93.0"
19) "description"
20) "Bike messenger from New York, Cam (Taylor Lautner) is the best on two wheels but is indebted to a gang of mafia criminals. When he runs into the unknown and sexy Nikky (Marie Avgeropoulos), Cam is immediately seduced by her and the exciting world of parkour."
21) "humor"
```

In Redis, an aggregate is a key-value pair



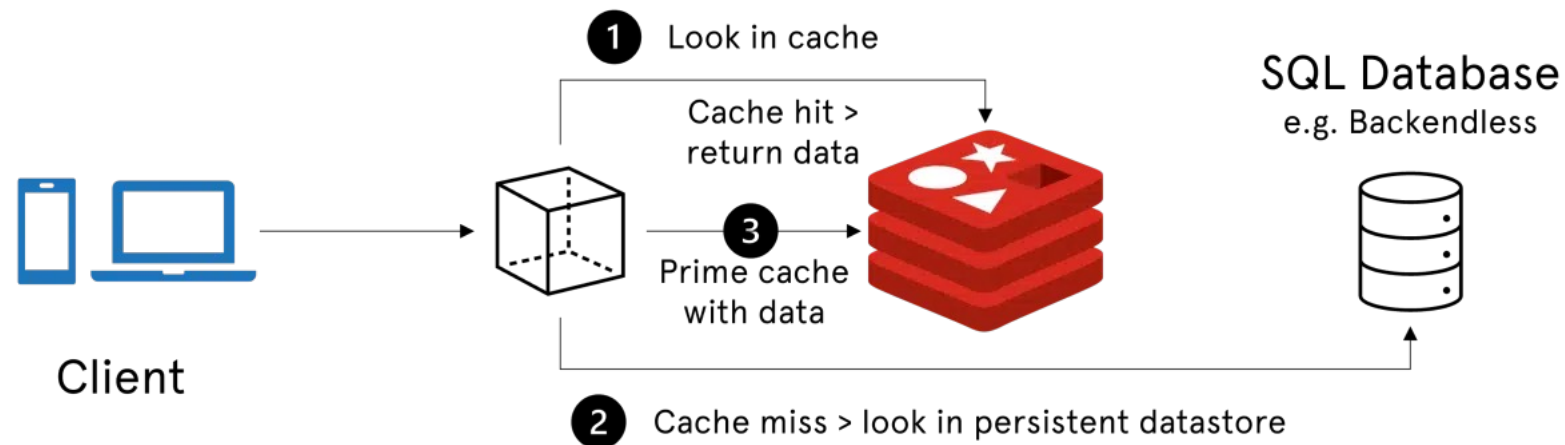
Introduction to Key-Value Storage

- ❖ Key-Value stores allow for fast access to data via key lookups
- ❖ They work like a huge hash map, where data is stored and retrieved by means of a key
- ❖ They are mainly used in ambits like caching data, and session management for users



Redis Typical Use Case

How Redis is typically used



How Redis Stores Data (1)

Redis, in contrast to traditional DBMS, exploits an in-memory storage to fastly retrieve data.

Each blob of data (value) can be of multiple types:

- Simple strings
- Hashes, conceptually similar to dictionaries
- Lists and sets (also sorted)



How Redis Stores Data (2)

```
127.0.0.1:6379> SET movie:99999 "hello"  
OK
```

```
127.0.0.1:6379> HSET movie:99999 title "Test"  
1  
127.0.0.1:6379> HGETALL movie:99999  
1) "title"  
2) "Test"  
127.0.0.1:6379> HSET movie:99999 genre "Comedy"  
1  
127.0.0.1:6379> HGETALL movie:99999  
1) "title"  
2) "Test"  
3) "genre"  
4) "Comedy"
```



How Redis Maintains Data (1)

Since Redis is an in-memory data storage, there must be a way to synchronize data to disk to prevent loss of information when the server is restarted.

Two main techniques are used to persist information: RDB and AOF.



How Redis Maintains Data (2)

- RDB (Redis Database File): data is stored on disk with a snapshot of the dataset at configured intervals. It is enabled by default

```
127.0.0.1:6379> CONFIG get save
1) "save"
2) "3600 1 300 100 60 10000"
```

- AOF (Append-Only File): every write information is stored on a log file, and it can be replayed in case of data loss

```
127.0.0.1:6379> CONFIG get appendonly
1) "appendonly"
2) "no"
```



How Redis Ensures Consistency (1)

Consistency is ensured natively, since Redis is single-threaded and runs in the basic setting on only one server instance.

Additionally, if we want to execute a sequence of commands without being interrupted by other clients' executions, Redis supports MULTI commands.



How Redis Ensures Consistency (2)

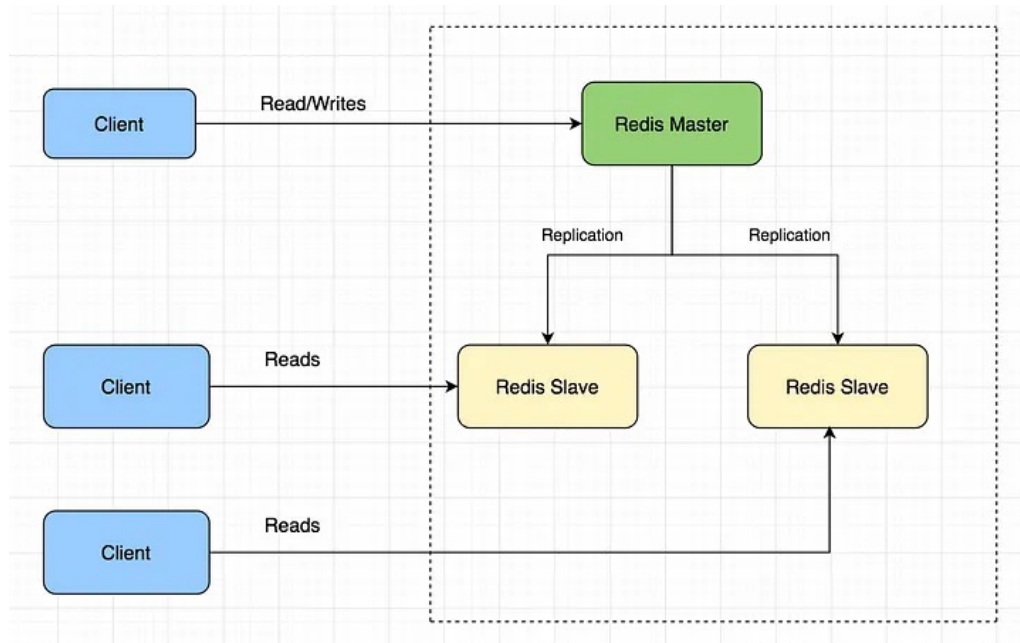
Nonetheless, when scaling horizontally with Redis Cluster or when increasing availability with Master-Replicas server, we may lose some degree of consistency.

This is consistent with what we have observed with the CAP theorem.



How Redis Ensures Availability (1)

When scaling horizontally, a server can be either run as a Master or a Replica (Slave):



In a typical setting, clients communicate with Master instances to write data, and the Master will **asynchronously** communicate with its replicas



How Redis Ensures Availability (2)

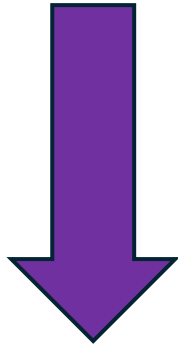
In the context of Availability, if one instance fails, we have two cases:

1. If a Replica Server fails, we simply create another replicating all data from the other Replica
2. If a Master Server fails, two approaches are possible
 1. We can add a new server as Redis Master
 2. We can 'promote' an existing Replica to the role of Master



How Redis Ensures Availability (3)

Since the Master Server asynchronously updates its replicas when a write request comes from a server, in case a failure occurs during the Master-Replica update, the new data is lost



No strong consistency guaranteed!



How Redis Performs Scaling (1)

Due to the need in Big Data Management to deal with large-sized information, Redis adopts a deployment topology called Redis Cluster that allows to scale horizontally.

With Redis Cluster, it is possible to automatically split data in multiple nodes, increasing the overall throughput of the system.



How Redis Performs Scaling (2)

Data is sharded through an hashing system, where every key is part of an hash slot computed by the modulo 16384 of the key.

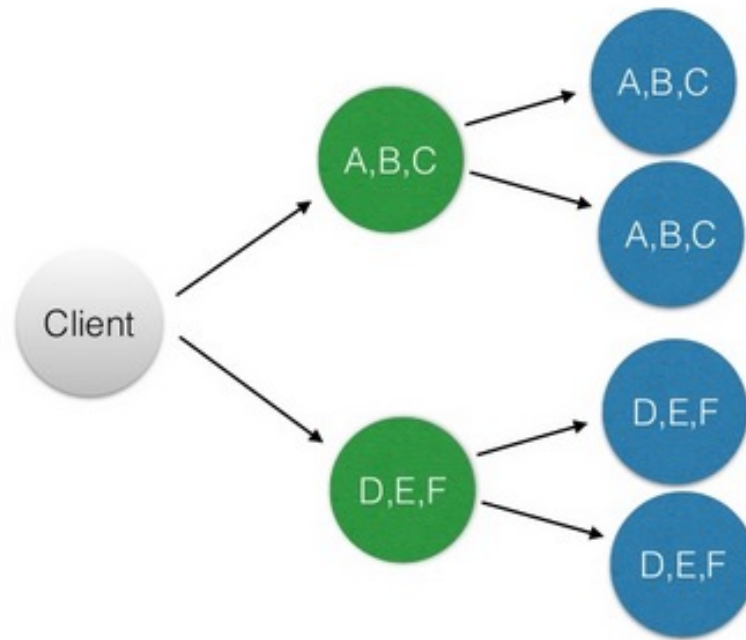
Finally, it is also possible to integrate Redis Cluster with the Master-replica model in order to increase the level of availability.



How Redis Performs Scaling (3)

Redis Cluster

Sharding and replication (asynchronous).



Redis indexing and queries (1)

Redis allows direct access to key-value pairs by knowing the key, which can be semantically meaningful (e.g. user:userId) with complexity $O(1)$.

However, when dealing with hash values, access to secondary key is also possible, by exploiting Redis additional data structures.



Redis indexing and queries (2)

There are two main types of indexes:

- Inverted index: used when indexing string fields inside hashes, for every term a specific list is maintained with the set of all documents containing this term
- Numeric index: a binary range tree where each numeric value is a node is maintained, so that also range operations can be performed



Introduction to Document-Based Storage

- ❖ Similarly to key-value stores, also document-based storages store and retrieve data by means of an ID
- ❖ However, they were born to support structures such as array and data types inside every document
- ❖ Thus, it is is natively possible to perform queries and aggregations on single fields



How MongoDB Stores Data (1)

In MongoDB, data records are stored as BSON documents

```
_id: ObjectId('67b114361b61cc4d0a2bdcf7')
title: "The Deciding Kiss"
year: 1918
cast: Array (2)
  0: "Edith Roberts"
  1: "Hallam Cooley"
genres: Array (1)
  0: "Comedy"
href: "The_Deciding_Kiss"
extract: "The Deciding Kiss is a 1918 American comedy film directed by Tod Brown..."
thumbnail: "https://upload.wikimedia.org/wikipedia/commons/thumb/c/cf/TheDecidingk..."
thumbnail_width: 320
thumbnail_height: 524
```

BSON is simply a binary representation of JSON documents, that extends some data types not available in JSON like dates, binary data, and also contains information like field length



How MongoDB Stores Data (2)

Documents are gathered together in collections, which are ideally equivalent to RDBMS tables.

In turn, a collection is part of a single database.

MongoDB documents are:

- Polymorphic: fields can change from document to document within a single collection
- Self-describing: No need to declare fields priorly, they are described on-the-fly when creating and updating documents



How MongoDB Stores Data (3)

Conversely to Redis, MongoDB stores data permanently on disk.

To ensure a greater level of data durability, however, it also exploits a system of checkpoints and data journal (similar to Redis AOF) which is also used during transactions to ensure consistency.



How MongoDB Ensures Consistency

Operations that affect single documents within a collection are natively consistent.

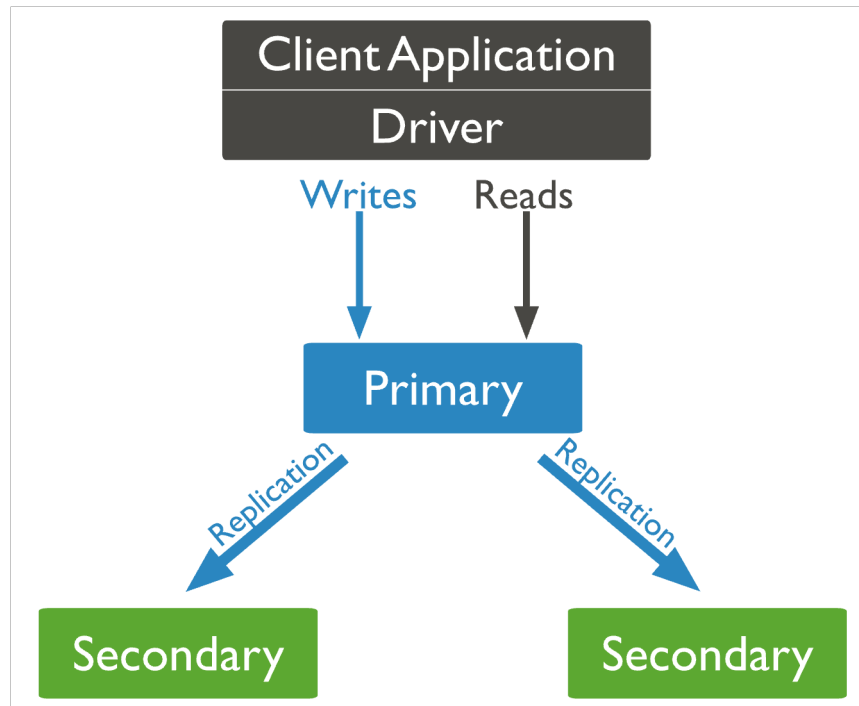
For cases when an operation updates multiple documents within one or more collections, MongoDB supports transactions.

Transactions can also be executed on multiple Shards or Replica Sets.



How MongoDB Ensures Availability (1)

Similarly to Redis, MongoDB provides increased data availability by replicating data inside a Replica Set



How MongoDB Ensures Availability (2)

The primary node receives all read and write operations, while the secondary nodes only perform replicate these operations asynchronously such that their data set reflects the primary one.

When a primary node fails, a secondary node can 'propose' himself as the new primary, and elections are held.

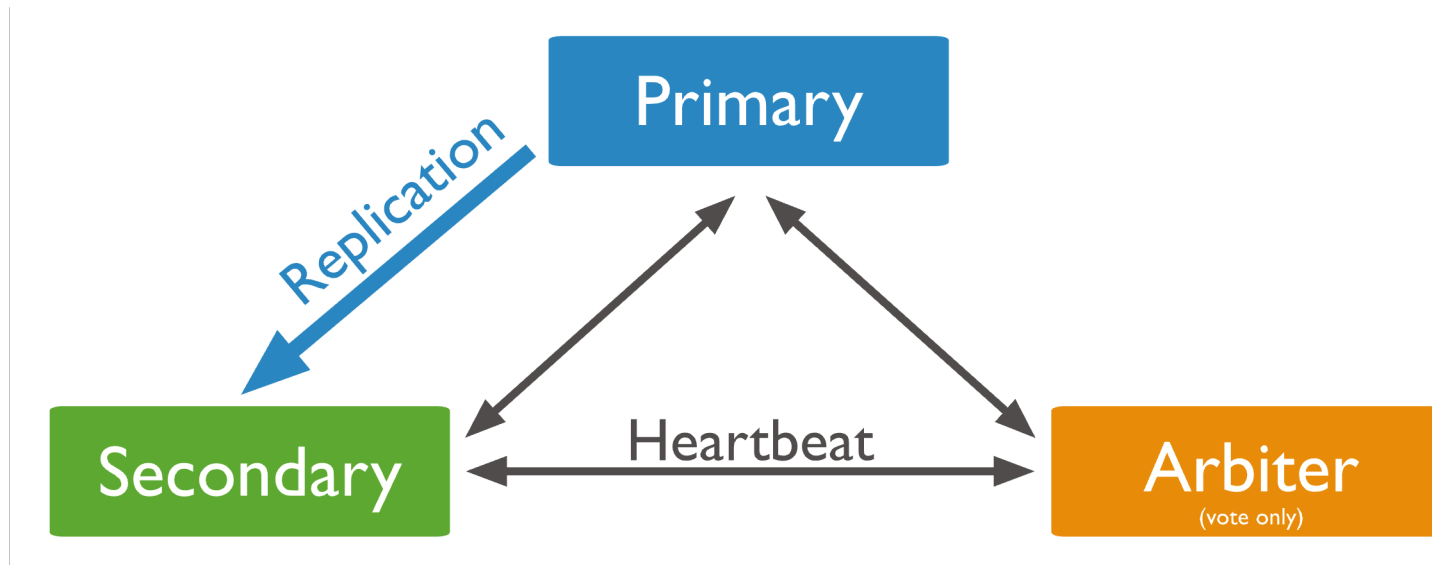
A special type of node called Arbiter can be added, that doesn't hold any data but can vote in an election.



How MongoDB Ensures Availability (3)

Every two seconds, each member of the Replica Set can send an 'heartbeat' (ping) to every other member.

If a member does not answer, it is marked as inaccessible



How MongoDB Performs Scaling (1)

Just like Redis, MongoDB was developed with the clear goal of handling massive datasets that power Big Data use cases.

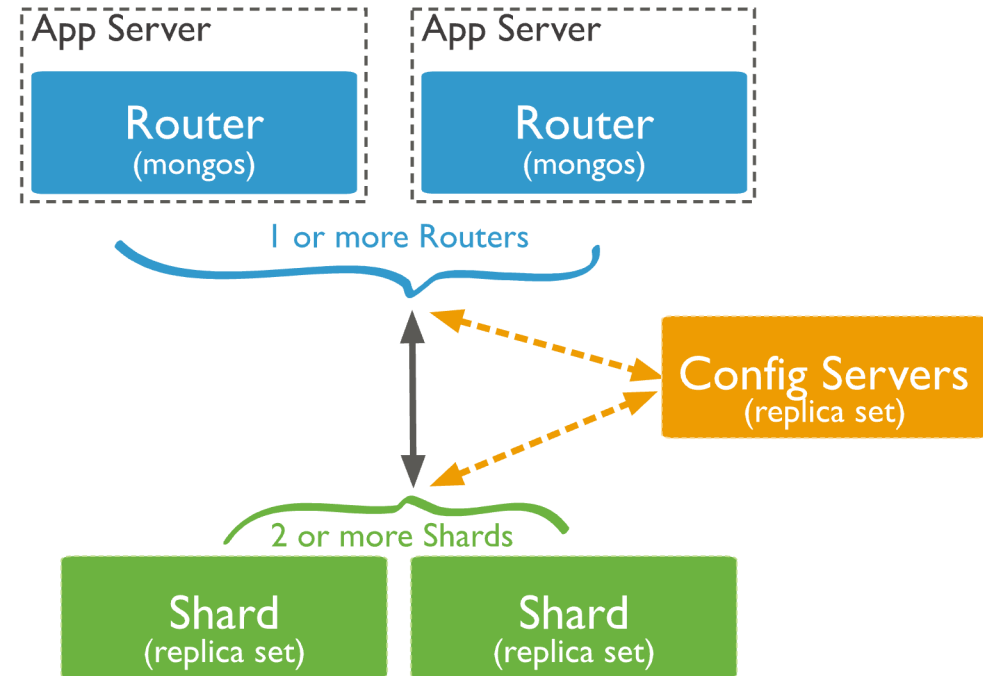
The same technical solution is implemented: horizontal scaling by means of sharding



How MongoDB Performs Scaling (2)

Inside a sharded cluster, three main components are located:

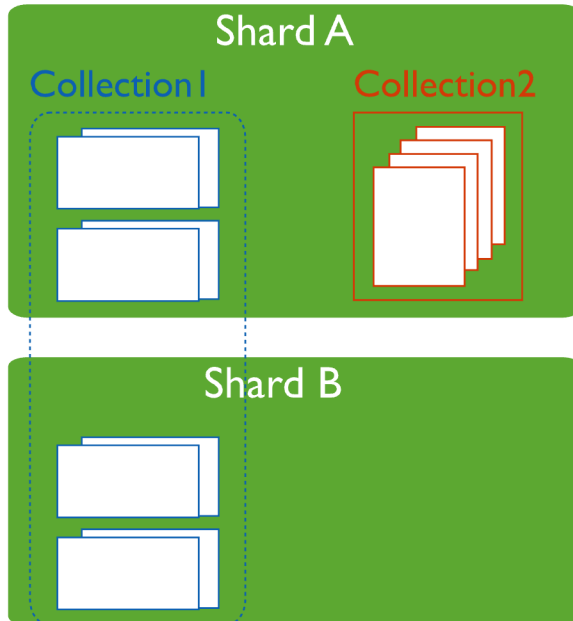
- Config servers: store metadata and settings for the cluster
- Mongos: Interface between application and shards, query router
- Shard: Actual Replica Set that contains data



How MongoDB Performs Scaling (3)

MongoDB uses shard keys to distribute the documents of a collection across shards.

A shard key is composed by one or more document fields which are declared when sharding a collection.



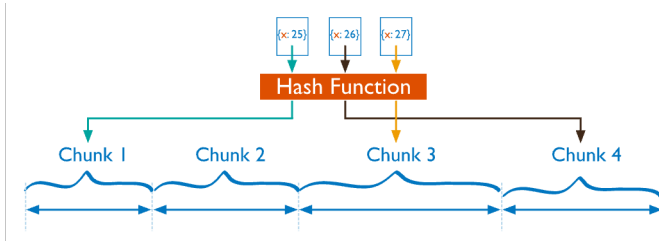
It is possible to have some sharded collections, whose content is distributed across partitions, while keeping unshared collections that can be located (entirely) on any shard.



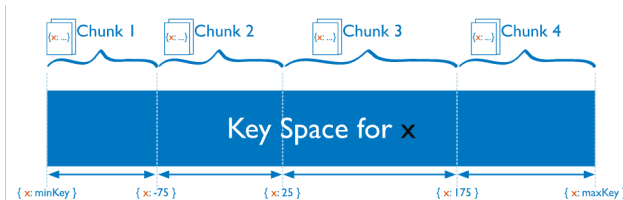
How MongoDB Performs Scaling (4)

MongoDB supports two different strategies for distributing data between shards:

- Hashed sharding: computed by hashing the shardkey



- Ranged sharding: dividing data based on ranges defined on the shard key values



Redis and MongoDB Recap (1)

	Redis	MongoDB
Data Model	Key-Value	BSON Documents
Data Storage	In Memory	On Disk
Consistency	By default + multi commands	By default + transactions supported (multi-collection ops)
Querying	Key lookups + optionally Redisearch, indexes specified priorly	Any field
Scalability	Yes, but harder to maintain with hashing	Yes, automatic data distribution
Availability	Possible, but no automatic failover detection	Natively supports automatic failover detection



Redis and MongoDB Recap (2)

- **Redis:** Best for **in-memory speed**, **caching**, **session management**, with **limited query support** unless using **RediSearch**
- **MongoDB:** Best for **flexible document storage**, **complex queries**, and **high write/read scalability** via **sharding**



...Thanks!

References:

- Theoretical notions: LSDM Course taught by prof. Antonella Poggi
- [Redis documentation](#)
- [MongoDB documentation](#)
- GitHub: [LSDM-Big-Data-Management](#)

