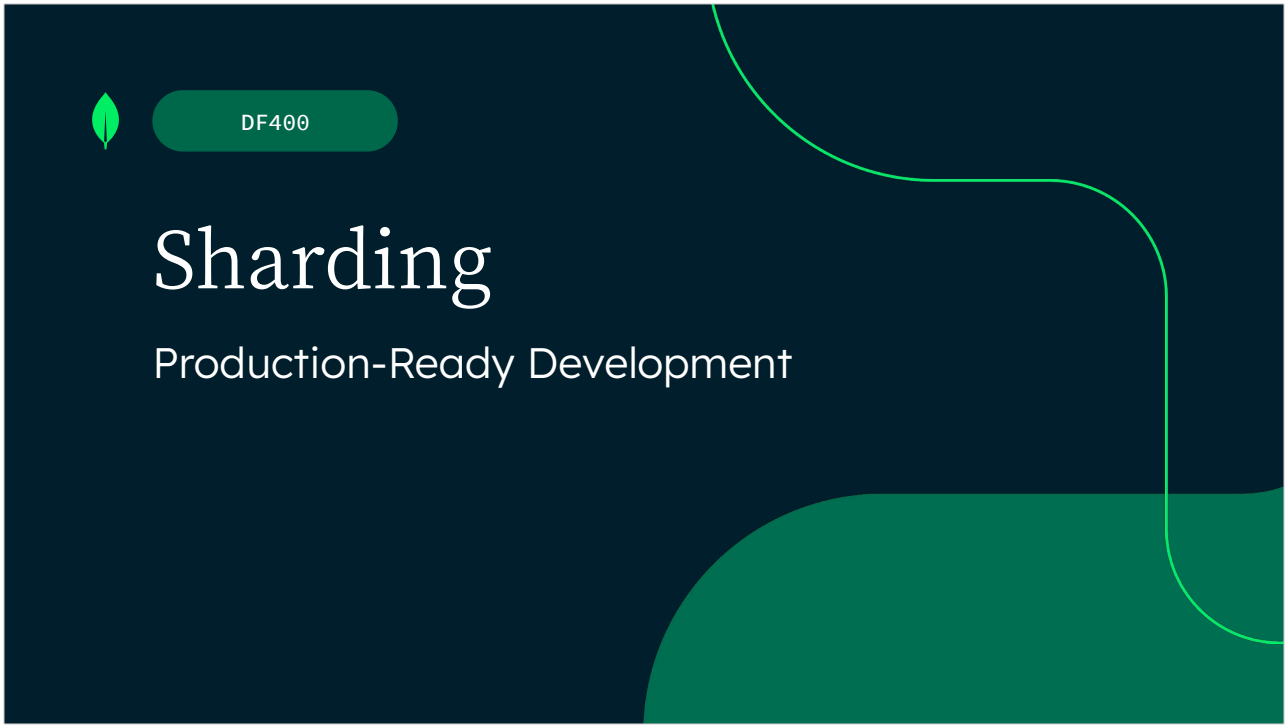


Release: 20230414

Topics we cover

Sharding

Horizontal vs. Vertical Scaling

Components of a Sharded Cluster

Shard Keys

How Sharding works

Sharding

Sharding is a mechanism for partitioning larger collections across multiple servers

- Enables bringing unlimited hardware to a scaling problem
- Allows parallel processing for some tasks
- Allows enforcement of geographical data location

Sharding is utilized for horizontal scaling.

Sharding vs. Replication

Replication is to ensure resilience

Sharding is about bringing more capability

Many baby ducks is resilience, some don't survive (HA) . Wolves form a pack of adults because a group is more capable than one, they divide up the task of hunting between them working together.

- In some NoSQL systems Scaling and High Availability (HA) use the same mechanism - In MongoDB, they do not
- Replication is used for High Availability
- Sharding is for Scaling Out large workloads.
- Sharding without HA is wrong
    - Increased risk of something breaking not reduced
    - Higher cost and time to repair

Resources used by a database

CPU Cycles	Document DBs often use less CPU as not joining
RAM	Acts as a cache and much faster access to data than Disk
	Ideally should hold all your frequently accessed data
Disk Reads and Writes	Provides long term storage but slowly
	Measured in: Disk capacity, throughput, Operations per second
Network	Relevant for data moving about

It is essential to understand how the database uses different resources.
What resource is it that is limiting performance, what one will run out first?

Vertical vs. Horizontal scaling

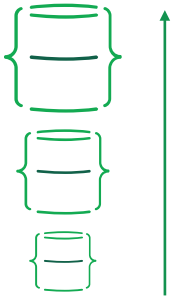
There is a scaling limit for a single server

With horizontal scaling, there is no limit

Vertical vs. Horizontal scaling

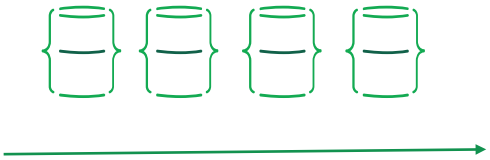
Vertical Scaling

Increase size of instances (RAM, CPU etc)



Horizontal Scaling

Add more instances (small or large)



Limitations of Vertical Scaling

Adding more hardware is called Vertical Scaling
Limits to how much machine's resources can be increased:

- Architecture: Max number of cores/RAM
- Provider: Cloud instance sizes
- Cost: Price isn't linear once things get large

- Vertical scaling has limits that often prevent the ability to scale easily
- Few places are happy to provision a server with 128 cores and 4TB of RAM

Case of study

In 2014 a fitness app company had a MySQL server with 750GB of RAM

- Expanding to 4TB of RAM as they grew was disproportionately expensive
- Decided to changed to 4 MongoDB shards, each with 512 GB
- Now they have expanded beyond this significantly

Example of how application growth can take advantage of horizontal scaling

When to shard?

Sharding is a solution to Big Data problems

- Don't need to shard until you have a sizeable problem
- Big data can be simply an aspect of many users
- The exact size differs based on the use case
- Vertical scaling is much easier up to a point

How to tell if you need to Shard

- Resource are maxed out (and you know why)
- Schema and code are already optimized
- No affordable update to the current server can resolve the issue
- Need to meet a backup restore time (RTO) target and need parallelism to do so

Sharding is required when the data exceeds the capacity of your system's RAM or Disk IOPs.

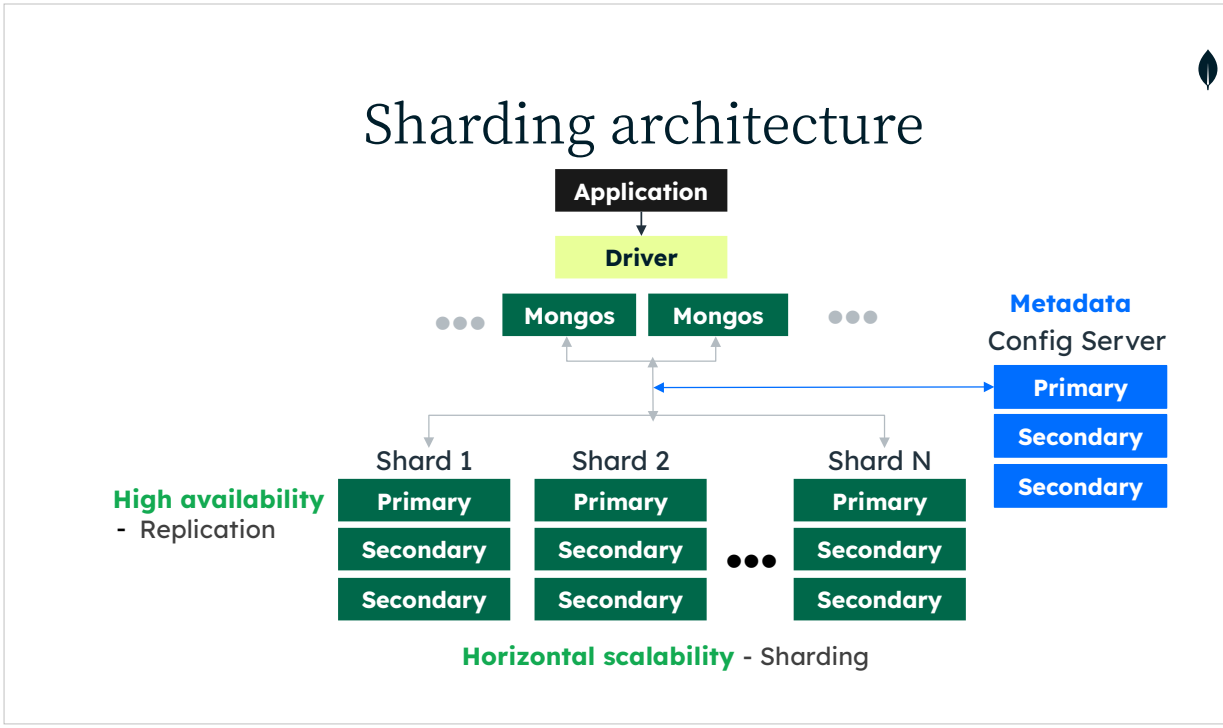
On a separate note, Atlas provides a hard limit of 4TB per shard (applicable for clusters which are upto M40 instances, higher level of instances have higher capacity).

Why understanding sharding now?

Might not need to shard now but schema design decisions taken today matter in sharding

Start planning for sharding on day one

It is important to understand the concept of sharding so that you can plan for future growth.

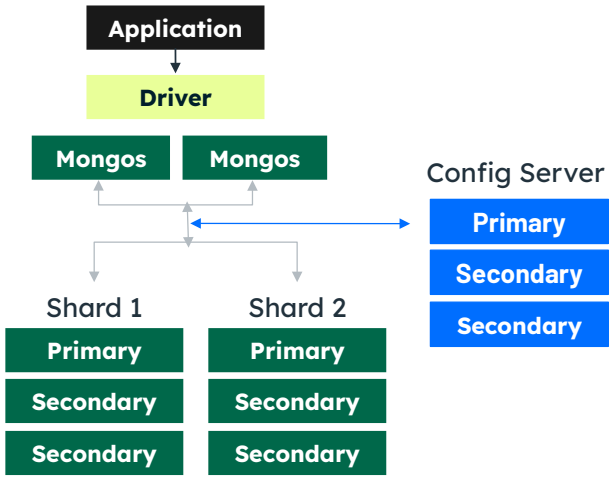


Sharding has a number of moving parts - each of which should be planned and sized. Sharding is about solving big problems - you can do a huge amount of processing before you need to go there so don't be afraid of the complexity - remember that you may be managing a 200 server cluster ultimately and that means you are doing something that has real world value.

Sharding uses more hardware

Sharding adds:

- A config (metadata) RS (3 nodes)
- Ideally at least 2 query routers
- Several Replica sets holding data

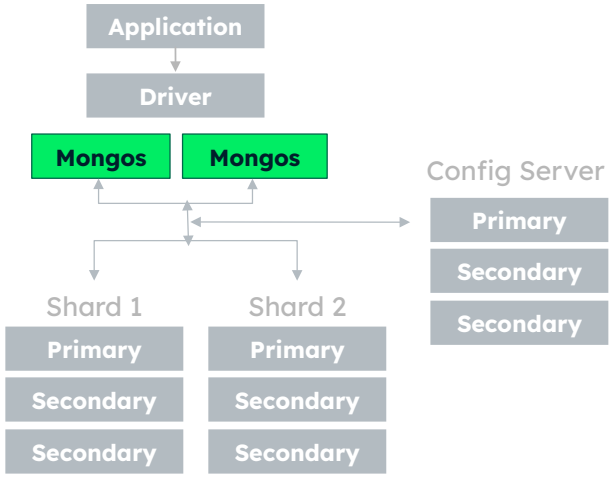


- The smallest production cluster should be 11 nodes.
- 2 Replica Sets to hold data
- 1 Replica set to hold metadata
- 2 Routers to avoid a single point of failure
- Sometimes these are combined - e.g., a router on a data server.

mongos

Proxy server you connect to in sharding

- Behaves like a MongoDB database but routes requests instead
- Only sends work where it needs to
- No local storage needed
- Good network connection needed



mongos routes requests (replaces mongod from standalone / replica set)

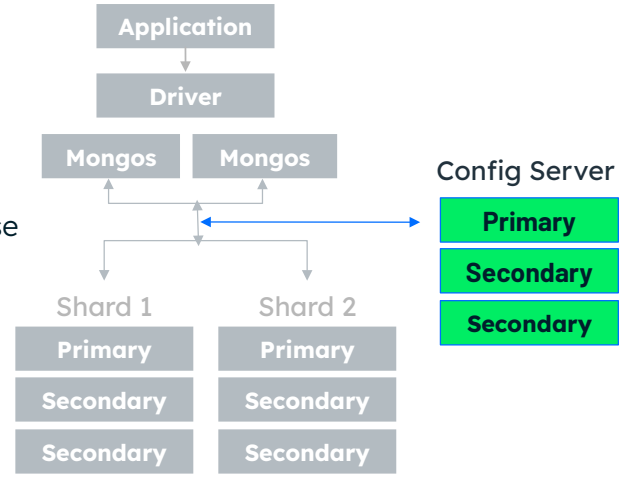
mongos will balance the chunks by monitoring chunk distribution across different shards and coordinate the chunk migration.

Configuration servers

Config Replica Set (Config server)

- Holds metadata about users, partitioning, and data locations
- Metadata store in 'config' database
- Accessible via mongos
- Does balancing of the chunks

Always a replica set - Why?



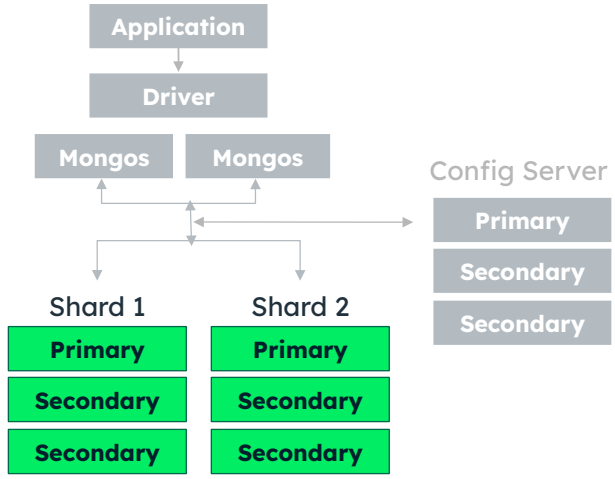
Configuration servers hold metadata about users, partitioning, location of ranges of data..

Shard servers

Each shard

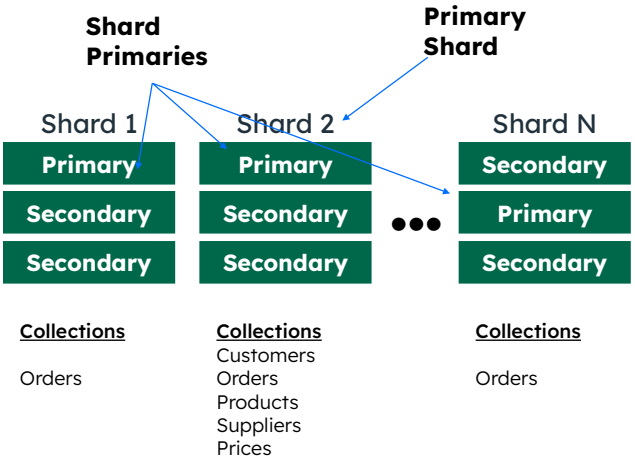
- Must be a replica set (RS)
- Holds a subset of your largest collections

One shard alone isn't useful, from two to thousands of shards are



- Each shard is a replica set.
- We do not want a shard to be unavailable; hence each shard must be a replica set.
- Each additional server statistically increases the chance of any one server failing, by using replica set, a server failing doesn't break the system.

Primary Shards vs. Shard Primaries



A Shard Primary is whatever node is the current Primary in the Replica Set representing that Shard.
The Confusingly similar term "Primary Shard" is the shard , for each database, designated when sharding was enabled to hold any unsharded collections in that database,
The shard primary is also the shard used for any other tasks that cannot be done across all the shards.

Shard keys

Shard key means two things

- Fields chosen to partition data
- Values on those fields in a given document

The term Shard key is used in two different but related ways.
The shard key of a collection is what the field names and order are for the partitioning.
For example, partition (shard) it by surname and county.

But in a document, the shard key is Jones and Orange - the value of the shard key.

Choosing Shard Keys

Select what fields to use to locate the data

Changes on shard key (based on version)

- Before 4.4, Cannot change the shard key afterwards
- Version 4.4+, Can add another field to refine the key
- Version 5.0+, Can change the shard key and reorganize the data

Rules for choosing a good shard key

- Included in most of your queries
- Reasonably high cardinality
- Ideally no more than 64MB of data to share a shard key
- Co-locates data you wish to retrieve together

A shard key will determine how your data is partitioned.
If the cardinality is too low or too many items have the same value then you will get 'Jumbo" chunks - which cannot be split and balanced.
In v6 the chunk can be up to 128MB

Choosing shard keys in practice

Choosing a shard key is not challenging but requires some thought
Only shard large collections

In large collections, users work with subsets of the data

- User data - shard by a user, for example: bank accounts or games
- Departmental data - shard by department or branch
- Where there are no clear subsets, shard for parallelism
  For instance, in an analytic data store, add a random value for the shard key

The choice of a shard key should be considered carefully! It cannot be changed without dropping and reloading your data.
Once it's done, it's hard to undo;

Sharded operations

When having a good shard key

- Most operations target a single shard or a few shards
- Individual users mostly hit a single shard
- If a shard is down, only some operations fail

More efficient than every query going to all shards

- Less work done
- Lower average latency

An operation that cannot target specific shards is called "broadcast"
If a shard is down - all operations targeting it fail completely

- All scatter gather queries fail
- Ideally, only a subset of users are affected by a shard being down

A good shard key will target few shards, preventing a “scatter gather”

Exercise - Choosing Shard Keys

What fields could be used as shard keys in the following scenarios and why?

- A web-based email service like Gmail
- A government database of businesses and their directors
- A stock database for a national chain of electronic stores
- A customer accessible product catalog like Amazon

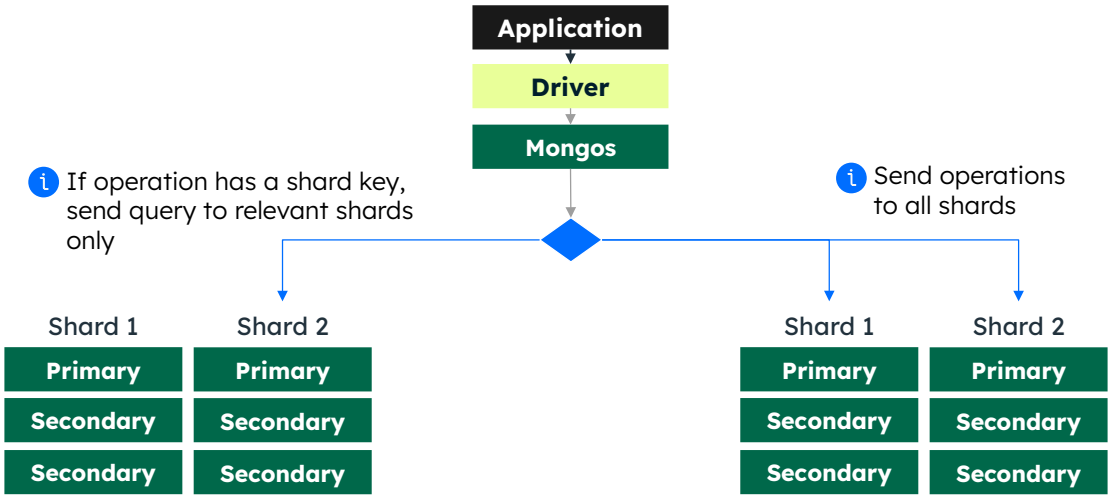
Steps to Shard a collection

1. Ensure the cluster is a sharded cluster - Check using `sh.status()`
2. Decide on the shard key
3. Configure the database for sharding:
   `sh.enableSharding("MyGameDB")`
4. Specify the shard key for the collection:
   `sh.shardCollection("MyGameDB.players", {playerid:1, gametime:1 })`
5. Check status again to verify chunks are moving - `sh.status()`

Ops Manager / Cloud Manager provides a GUI option to do this

If your collection already exists and has data then you will also need to create an index which beings with (or is) the shard key before you can run shardCollection. MongoDB needs this index to efficiently find and move chunks.

Sharding- distributing operations



- The query is sent to mongos by the driver
- IF query contains shard key (or at least first field of the key)
  - Lookup where data with that key is stored
  - Send query to those replica sets to run as normal
  - When results come back stream to the client.

  ELSE
  - Send query to all servers.
  - When results come back stream to the client.
- Concepts regarding reading from secondaries and read concerns still apply and the sharded replica sets are queried appropriately.
- If sorting required, do merge-sort of results as they come back

How config metadata caching works

- Driver sends operation to **mongos**
- **mongos** has a cached copy of config data which is versioned and uses this to target relevant shards with the operation
- The operation is sent along with the version of the config known by mongos

If **mongos** cache of config is an older version than the RS's knows

- Replica Set ignores operation and tells mongos to refresh the cache
- mongos does so, then re-sends operation to the correct servers

mongos doesn't need to ask the config servers every time they have a config 'version' which they send to the replica set with the write (or read) - the replica set know the most up to date config version and will tell the mongos if it needs to refresh its metadata.

Chunks

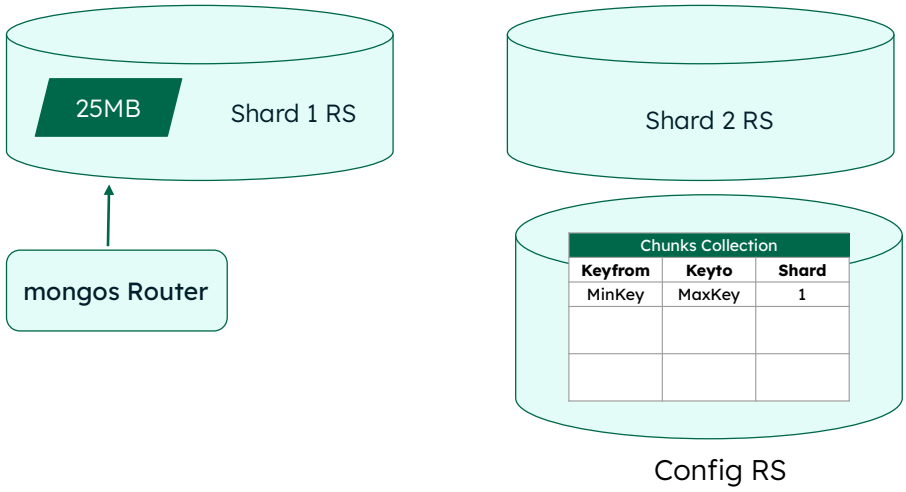
A Chunk refers to all documents where the shard key is in a given range of values

- Each Chunk exists on a single shard
- If a shard key falls into the range, then it is set to be in the Chunk
- Data in each range is always inclusive of its lower boundary and exclusive of its upper boundary
- The reads/writes are routed to a shard hosting the specific Chunk

Sharded collection data is stored in chunks.

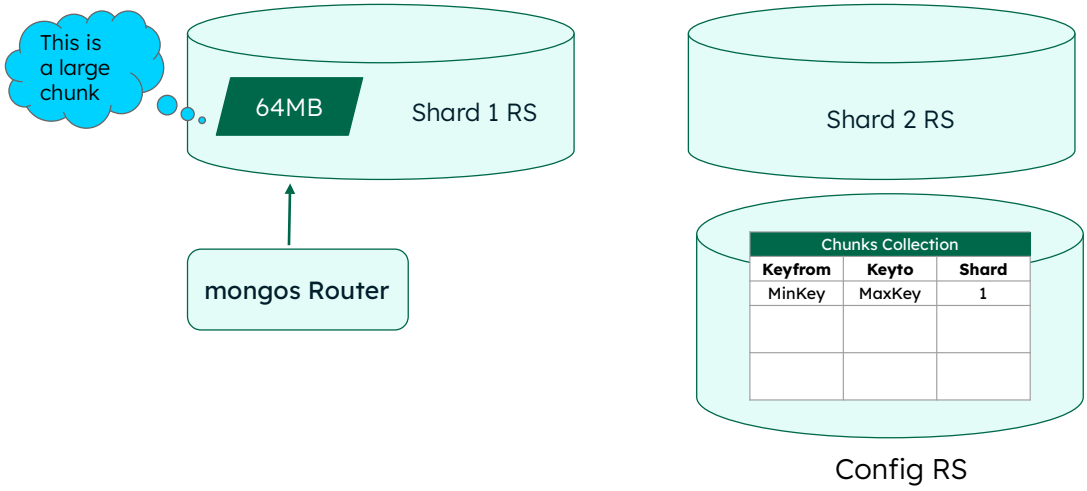
For example, Customer number 667726 might be in a chunk going from 660000 to 670000, and that chunk is stored on shard 8. Inserts, updates, or queries for that value will be sent just to shard 8.

Sharding in Slow Motion



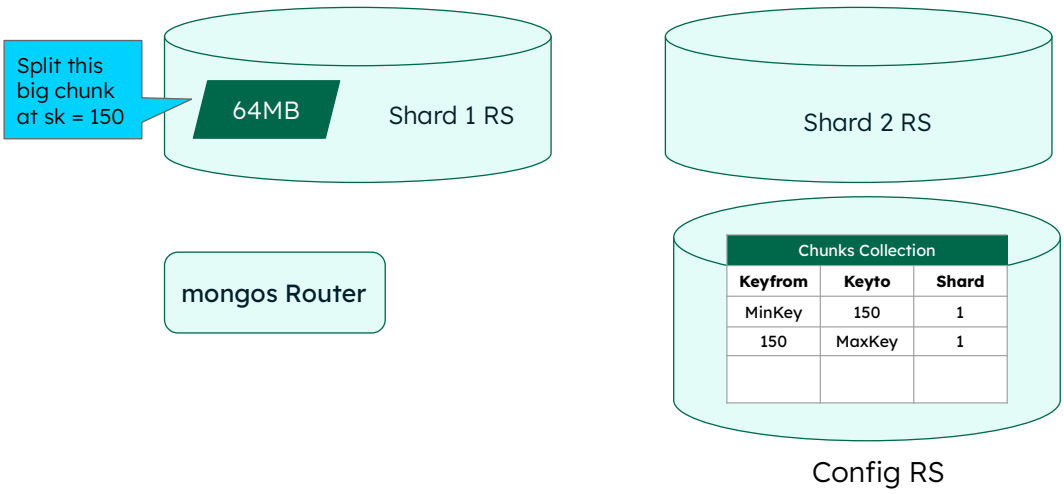
- With a newly sharded collection, it contains just one chunk from MinKey to MaxKey, which will hold all documents
- Therefore all writes initially go to a single shard.

Sharding in Slow Motion



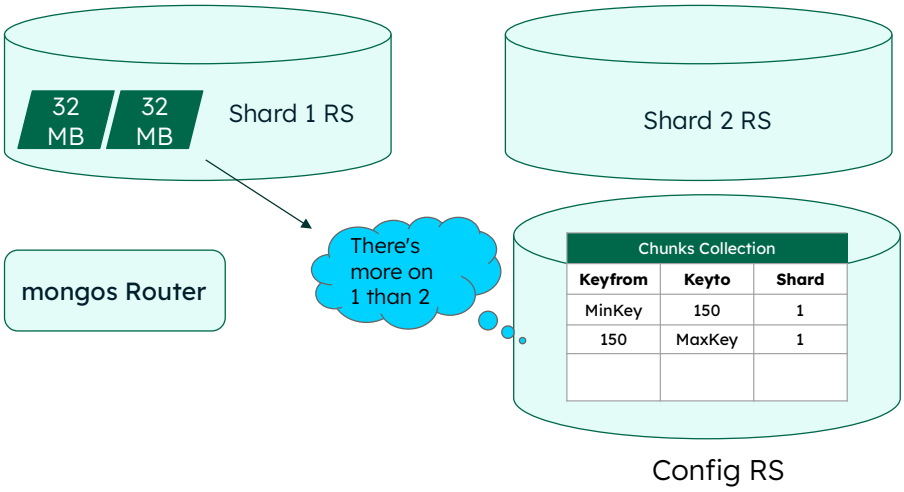
- The shard will realize at some point that a chunk has just written too is getting large
- We define what this is, but we want chunks to be small enough to move them easily. So by default, we want them to be less than 64MB (In v6, it will be 128MB)

Sharding in Slow Motion

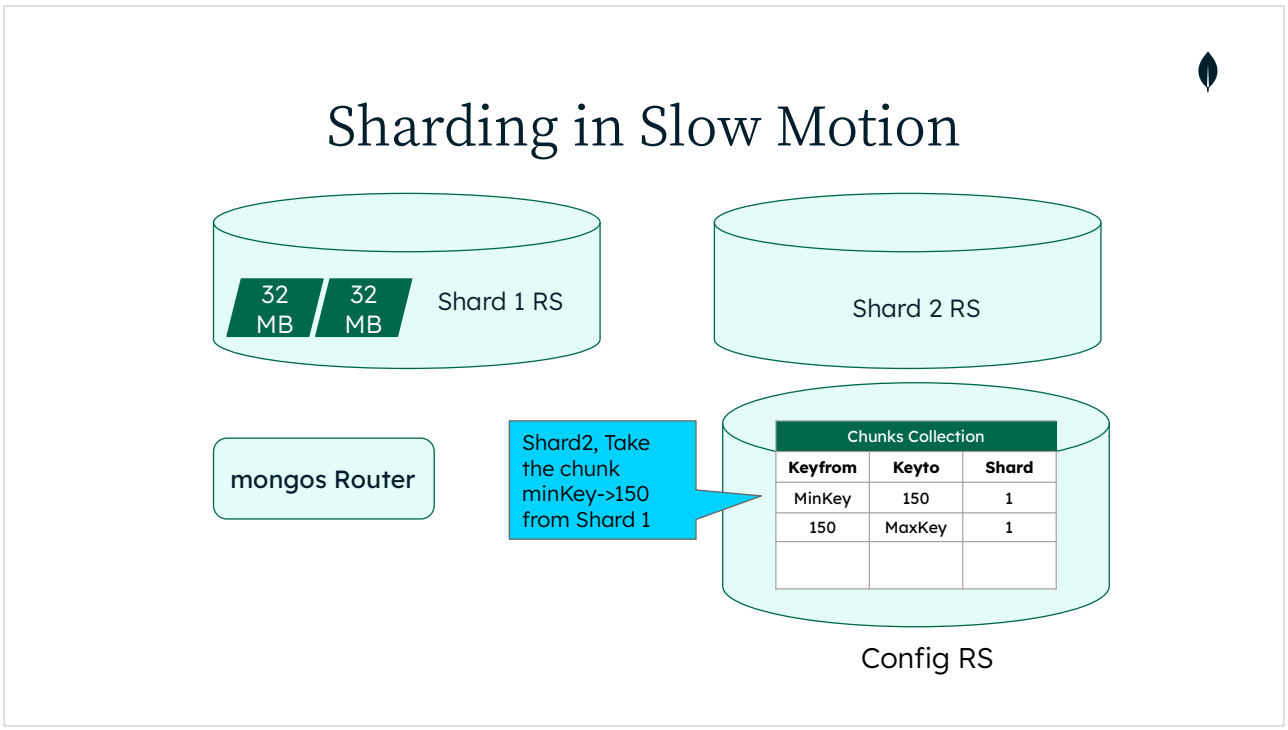


- The shard will work out where a good point to split it is and tell the config server this is a good change
- The config server will update its metadata (also, the routers will download this change) - no data moved during this auto splitting process.

Sharding in Slow Motion

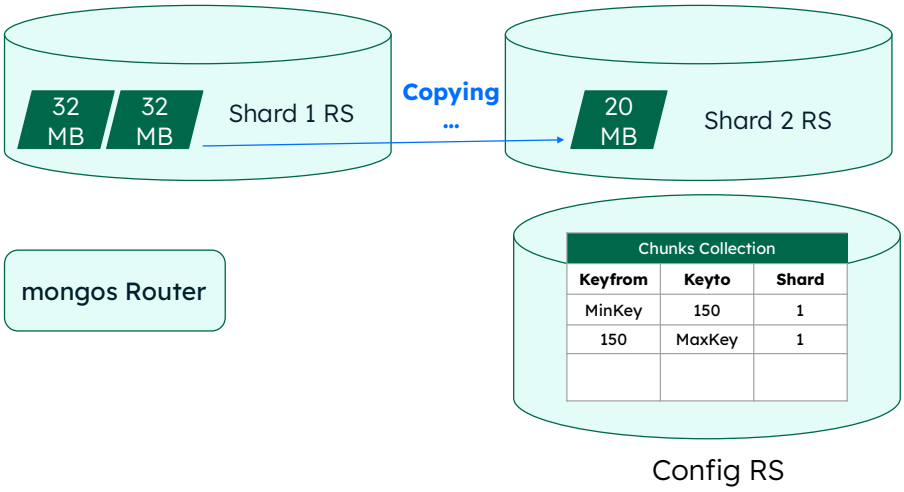


- At this point, some code called the balancer notices all is not right.



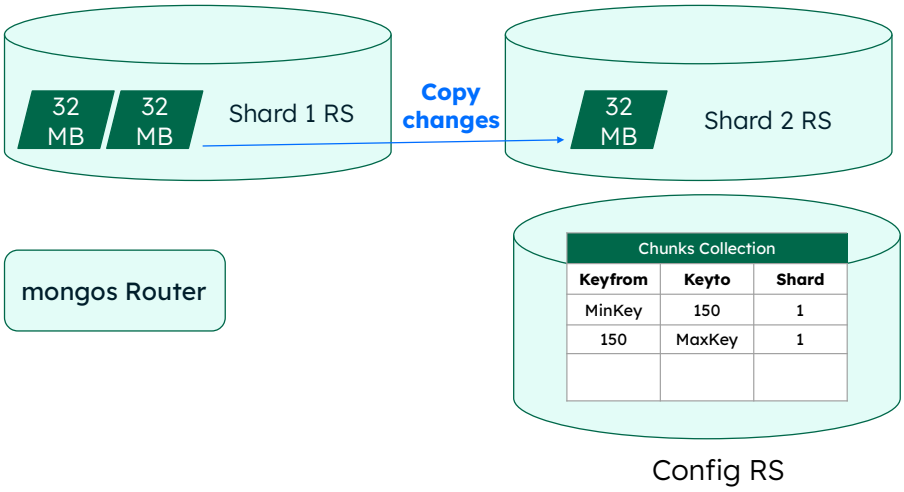
Instruction to initiate moving chunk from Shard 1 to Shard 2.

Sharding in Slow Motion

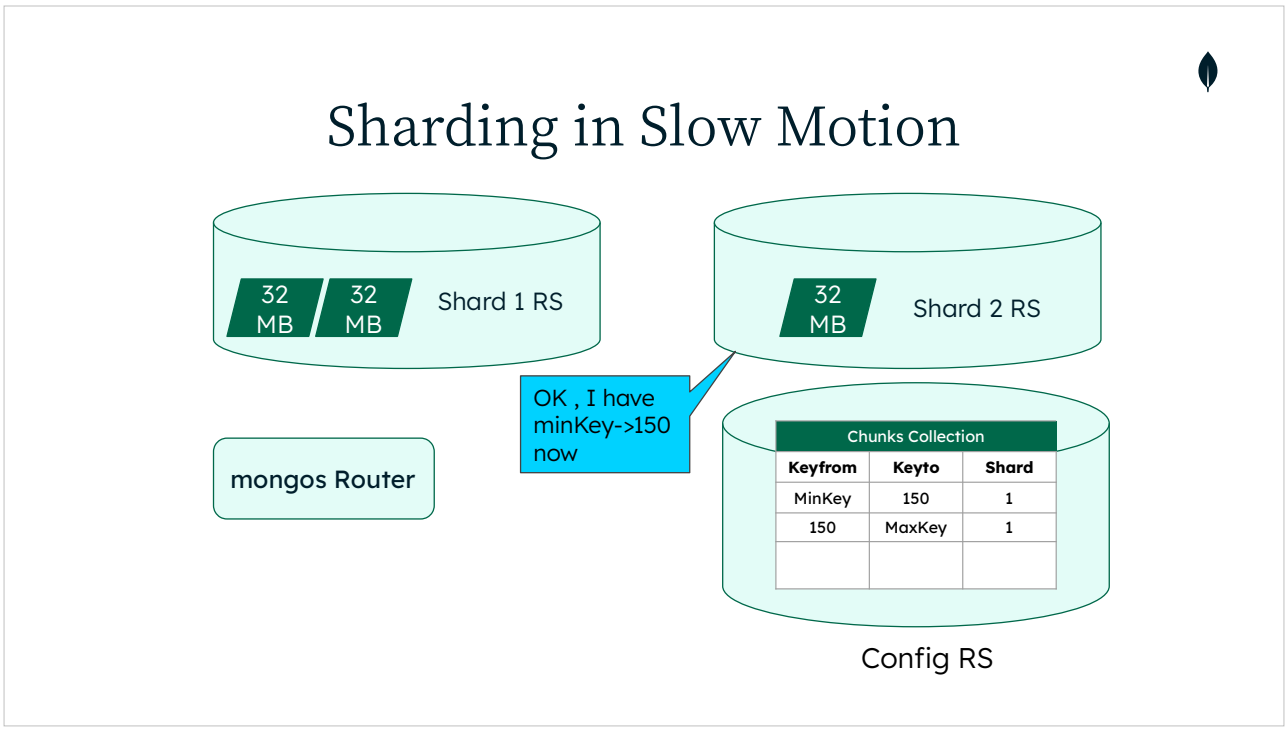


Shard 2 will begin copying the chunk from Shard 1

Sharding in Slow Motion

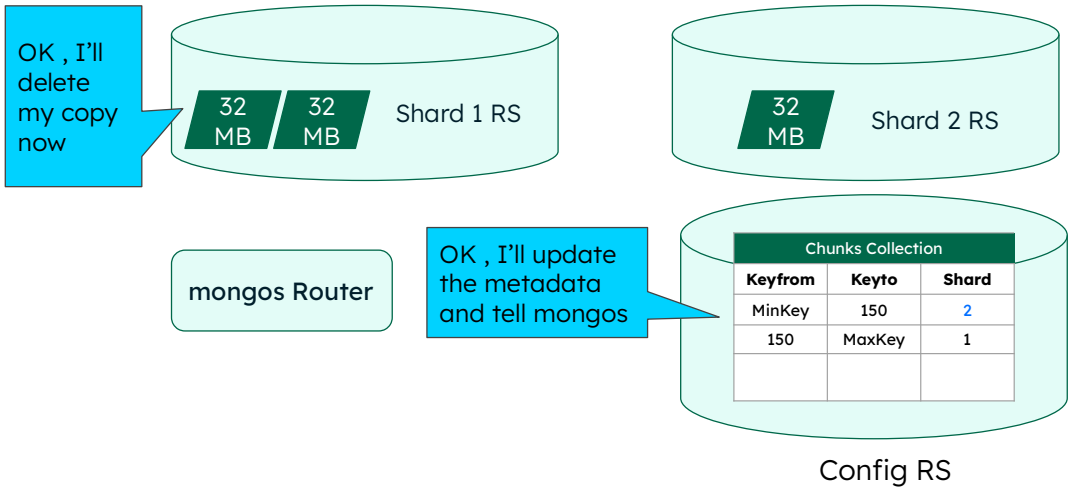


Shard 2 will eventually have the full chunk from Shard 1



Shard 2 will confirm it has the full chunk.

Sharding in Slow Motion



- Shard 1 will delete the chunk that has been copied
- Config server will update its metadata

Presplitting before loading

Presplit the data to avoid the initial shuffling via the first shard

Split chunks before they have any data in - Two ways:

- Balancer can move the empty chunks
- Explicitly move chunks - may be faster

Can speed up a bulk load of data

One thing we can do is, before loading any data is manually split the original chunk by specifying where it should be split.
Doing this generally requires us to have an idea of the range of values in the shard key.
Once we have split the original chunk into smaller (still empty - just logical) chunks we can also move them.
Moving an empty chunk takes very little time as there is no data to copy, it's just a metadata change.
This is done with the commands splitChunk and moveChunk
We only need to create one chunk per shard, especially if we move them manually.
A chunk cannot move BACK to a shard it came from for 15 minutes - so be careful when manually moving not to do this - the command hangs for a 15 minute timeout.
Best to split first then move all the chunks and do not move a chunk to the shard it's already on.

Sharding Pitfalls

Balance this is not very efficient

- Each chunk that moves writes three times (write on 1, write on 2, delete on 1)
- Getting the deltas takes time in a busy system (more reads and writes)
- Once chunks are distributed, if writes hit random chunks then is good - self balanced
- First balance can take a while

What if we are inserting and shard key is increasing?

- All writes go to same shard `{ sk: x } ->{ sk: maxKey }`
- 50% of chunks are moved as every other chunk gets copied to other shard - 3X writes!
- Shard by the MD5 hash of the shard key instead for random distribution

ObjectIDs start with a timestamp and are the default _id value in MongoDB. Each will mostly be just a little bigger than a previous one.
So if you shard by the default _id each write will go to the same chunk as the previous one, and the balancer will be moving stuff
to try and keep the chunks balanced. For two shards 50% of document will be written (To one shard) then read , rewritten to the second shard and deleted from the first.
It is important to not have incrementing shard keys likes this but to ensure they have a good distribution over your shards.
There are a number of techniques to achieve this but it's a common mistake.

The story of Alex

Case study example

Alex learns MongoDB

Alex wants to see if MongoDB is Fast and writes something in the shell to test

```
while(true){
    db.test.insertOne({txt:"Is this fast?"})
}
```

Alex is impressed by 3,000 inserts per second

Alex scales out

Alex wants to see how well MongoDB scales:

- Creates a 3 shard cluster
- Shards on _id

Running the test again:

- Writing only to Shard 1
- Balancer reads and writes to the other shards

Now only gets 2,500 writes per second because they are underloading one shard

Alex is confused

Alex tries YCSB

Someone explains to Alex that they need:

- A multi-threaded load generator
- Something faster than Javascript
- Load data in batches, not one document at a time

Alex tries again with YCSB (NoSQL Benchmark tool)
This time a single Replica set manages 40,000 per second
Alex is amazed!

However, the three shards manage just 20,000 between them!

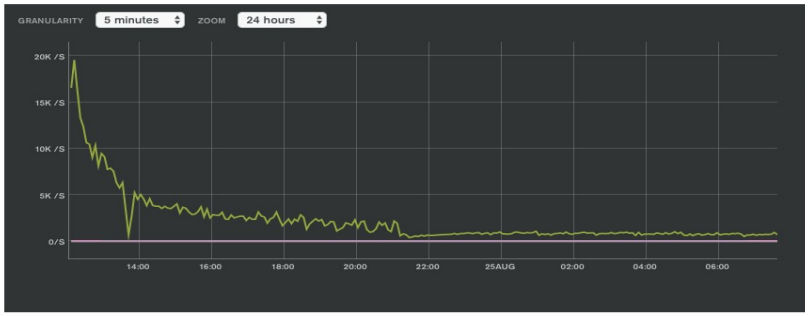
- This happens if all writes go to one chunk due to an increasing shard key
- Someone in a forum recommends using a hashed shard key.

YCSB - the Yahoo Cloud Serving Benchmark is actually a benchmark for Key/Value capability. Written in 2010 it was never a good way to measure performance of a document store as it misses a lot of the capabilities and is in no way optimised for the model.
It's frequency used to compare different technologies to each other where no valid comparison can be made.

Alex tries hash

Alex uses a hashed shard key for _id

- Now writes are spread evenly across all the shards
- First initial minutes, 120,000 writes/s
- Then, it starts to drop off and gets slower



Case study example

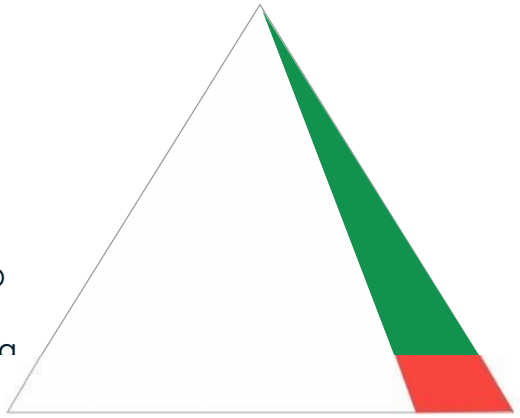
- This is from a customer - who asked for help because their data loading did not keep a constant speed even over a few hours.
- Between 12:00 and 21:15, the rate dropped as the amount of disk i/o increased.
- At approx 21:15, the index size grew larger than RAM, at which point it got very slow. This is impacted by disk speed - faster SSDs aren't as bad.

Downsides of hashed indexes

Indexes are BTrees with internal and leaf nodes

- White: data can be on disk
- Green: frequently read, some writes, should be in RAM
- Red: frequently written, needs Disk IO

With an unhashed index an Increasing key goes to the right of the tree



Case study example

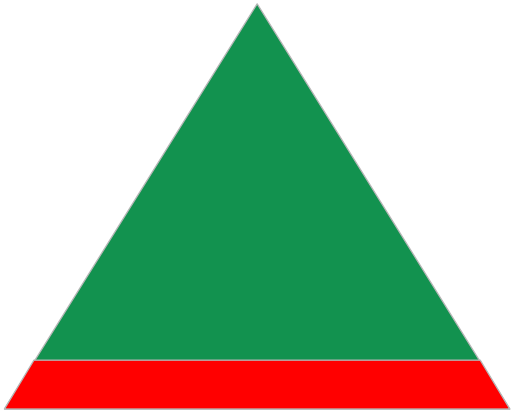
Downsides of hashed indexes

A random key (hashed value) could go anywhere in the index

The whole index is being accessed continuously, so it needs RAM

The maximum number of disk block are being dirtied

Simple hashed shard keys are not a good idea at scale



This also applies to other random values like GUIDs in indexes

Common sharding challenges

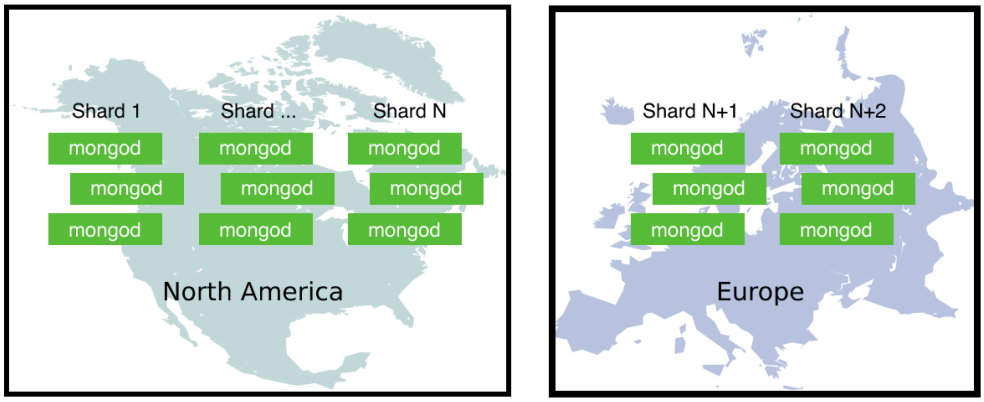
Data may be badly distributed

- The cluster eventually balances volume as long as there is a reasonable cardinality
- Can get 'Hot' shards doing a lot work relative to others
- Hashed sharding avoids hot shards but takes more resources overall
- The shard key may be a random value (e.g. MS GUID) taking too many resources

Scaling out a cluster can temporarily take a lot of resources

- Running out of resources implies a need to scale out!
- Easier to add more shards, but those are initially empty
- The balancer is responsible for copying data over to them
- Can take days or weeks as lots network and I/O !

Zone based sharding



Zone-based Sharding provides Instruction to the balancer where to store specific ranges of data. It is a way of controlling what the balancer does.

It is used for two things:

1. Providing better hardware for 'more important' data
2. Keeping data on specific servers for example in specific countries (Regulatory requirements)

Remember, you cannot write to a secondary. So how do you keep US data in the US and European data in europe but it appears as a single collection.
You have multiple shards, sharded by location. One for the USA one for Europe for example. Each is hosted in their own region.
The shard key determines if data should be stored in the US or Europe. You might have a 'Secondary' in the other region to make querying quicker.
.
Supported through the Atlas GUI for ease of setup (Screenshot)

Sharding for Parallelism

Unusual use case but powerful when used

- Reads sent to all shards when running aggregations and queries
- Bad for OLTP / Normal queries
- For a small number of analytics queries, the more CPUs the better
- Sometimes we run many shards on one server! (Microsharding)
  This works where:
  - All data can fit in RAM - Disk is not a bottleneck
  - Small number of power users - more CPUs than users
  - Write aggregations that can use parallelism

Clarification on (b) more CPUs than users:

- The query engine doesn't parallelize a single query, so if a query is perform on a non-sharded collection, it will effectively use one CPU core out of potentially many.
- If a collection is queried at a high rate, using microsharding effectively allows users to use more than one core for one query, because the query is split between multiple shards (assuming it's not targeted to a single shard in the first place).

Quiz Time!

#1. Which of these are reasons to use sharding?

A	Remove limitations relating to hardware size
B	Make it easy to delete a subset of the data
C	Enable subsets of the database to be hosted in specific geographic locations
D	Increase uptime of a system and High availability
E	Allow some tasks to run more quickly due to parallelisation

Answer in the next slide.

#1. Which of these are reasons to use sharding?

Remove limitations relating to hardware size

Make it easy to delete a subset of the data

Enable subsets of the database to be hosted in specific geographic locations

Increase uptime of a system and High availability

Allow some tasks to run more quickly due to parallelisation

Answer: A, C, E

#2. Which of these are attributes of a well chosen Shard Key?

A	Has a low cardinality
B	Keeps commonly accessed data on a single shard
C	Limits the impact on users of a particular shard, when it is down
D	Distributes all newly inserted documents across shards eventually
E	Targets most queries to a subset of shards

Answer in the next slide.

#2. Which of these are attributes of a well chosen Shard Key?

A	Has a low cardinality
B	Keeps commonly accessed data on a single shard
C	Limits the impact on users of a particular shard, when it is down
D	Distributes all newly inserted documents across shards eventually
E	Targets most queries to a subset of shards

Answer: C, D, E

#3. What are the things to do when loading a large data set into MongoDB.

A	Presplit and move chunks in the collection	B	Ensure the data loader is not a bottleneck	C	Delete all existing data in the system
D	Disable replication in the system	E	Ensure to load with majority writes enabled		

Answer in the next slide.

#3. What are the things to do when loading a large data set into MongoDB.

A	Presplit and move chunks in the collection
B	Ensure the data loader is not a bottleneck
C	Delete all existing data in the system
D	Disable replication in the system
E	Ensure to load with majority writes enabled

Answer: A, B, E

Recap

Sharding is a mechanism of partitioning large collections across multiple servers

Choose the shard key well is important

Atlas provides a hard limit of 4TB per shard (M40).

Planning for Sharding early can help a lot when need it

Many people struggle with Sharding due to not understanding it

Can also shard for geographic reasons