

Student: ضياء حنا

Teacher: محمد بشار دسوقي

Date: 19/4/2026

Mongodb sharding and replication

April 19 2026

This file is divided into two parts one for answering the lab theory questions and the second part is for lab work

Part 1 (Theory)

Question 1:

الجزء النظري

1. ما هي استراتيجيات تقسيم البيانات لتحقيق قاعدة بيانات موزعة؟ وضح حالات الاستخدام التي تناسب كل استراتيجية؟

Answer from google MongoDB website and use cases are from chatGPT:

1. Horizontal Partitioning (Sharding)

In this strategy, the table is split by rows. Every partition has the same schema (columns) but contains a different subset of the data.

For example, a "Users" table with 1 million rows is split into 10 partitions, each containing 100,000 unique rows.

Use Cases:

1. Massive datasets that exceed the storage capacity of a single node.
2. High-traffic applications (social media, e-commerce) where read/write operations need to be distributed across multiple servers.

2. Vertical Partitioning

In this strategy, the table is split by columns. Different columns of the same table are stored in different partitions.

A "User" table might be split so that "ID, Name, and Email" are in one partition, while "Profile Picture, Bio, and History" (heavy or rarely accessed data) are in another.

Use Cases:

- 1.Optimizing performance by separating frequently accessed columns from "heavy" columns (like BLOBs or large text).
- 2.Security requirements where sensitive data (e.g., credit card info) needs to be stored in a more restricted, encrypted partition.

3. Range Partitioning

Data is partitioned based on a continuous range of values of a specific key (the Partition Key).

Partitioning by Date (e.g., 2022 data in Node A, 2023 data in Node B) or by Alphabetical order (A-M in Node A, N-Z in Node B).

Use Cases:

1. Time-series data: Monitoring logs, financial transactions, or sensor data where queries often ask for a specific time range.
2. Reporting: When users frequently run queries like "Show all orders from last month."

4. Hash Partitioning

A hash function is applied to a partition key (like `user_id`) to determine which partition the data belongs to.

$\text{Hash}(\text{user_id}) \% \text{number_of_nodes}$. This ensures a mathematically random but deterministic distribution.

Use Cases:

1. Load Balancing: When you want to ensure data is distributed uniformly across all nodes to avoid "hotspots" (where one node is busier than others).

2. Systems where range queries are rare, and most lookups are based on a specific ID (Point Lookups).

5. List Partitioning

Data is partitioned based on a predefined list of discrete values.

You define which values go to which partition. For example:

Region = ('UK', 'Germany', 'France') → Partition A

Region = ('USA', 'Canada') → Partition B

Use Cases:

1. Geographic distribution: Storing data physically closer to the users in that region (Data Sovereignty/GDPR compliance).
2. Business logic grouping (e.g., partitioning by Department or Status).

6. Composite Partitioning (Hybrid)

This strategy combines two or more of the methods mentioned above.

How it works: First, partition the data by Range (e.g., by Year), and then sub-partition each range by Hash (e.g., by User ID).

Use Cases:

1. Extremely large-scale systems (like BigTable or Cassandra) where a single strategy isn't enough to manage the volume and variety of queries.

Question 2:

2. في حال وجود عدد كبير من الاستعلامات على جزء من البيانات (مثلا الاستعلام عن كتاب معين، في حال كانت قاعدة البيانات تخزن كتب) ما هو التصميم المناسب لمنع الاختناق واستيعاب هذا العدد الكبير من الاستعلامات؟

When a specific piece of data (like a "bestseller" book) receives a lot of queries, it creates what is known as a "Hotspot" or "Hot Key" problem. We talked about this in our previous report. Check it out. To prevent bottlenecks and ensure the system can handle this high traffic, we should consider the following:

1. Read Replicas(most important)

The most effective way to handle high read volume in a database is to use Read Replicas.

All the queries for that specific book can be spread across 5, 10, or even 50 different replica servers instead of hitting just one. This increases your "Read Throughput" linearly.

2. In-Memory Caching

Before the query even reaches the database, we could use a caching layer like Redis or Memcached.

When a user searches for the popular book, the system first checks the cache. If it's there (a "Cache Hit"), it returns the data immediately from RAM.

3. Consistent Hashing with Virtual Nodes

One implementation of consistent hashing could solve the hotkey problem as we talked about in the previous report

It allows for better distribution and makes it easier to add "Virtual Nodes" to handle the load of a specific range of data that has become "hot."

4. Content Delivery Network (CDN)

If the "query" involves not just data but also static content (like the book's cover image, a PDF preview, or a long description).

We can use a CDN (like Cloudflare or Akamai) to cache the book's details at the "edge" (servers physically closer to the users).

5. Query Results Materialization(You tried to remember the name of this in class)

If the query for the book is complex (e.g., involving joins, reviews, and ratings).

Use a Materialized View or a pre-computed summary table.

Part 2 (lab homework)

Step 1: Understanding the Homework Requirements

The goal is to build a distributed movie system with two distinct sharding strategies:

1. Movies Collection (Range-Based Sharding):

Shard Key: title (alphabetical).

Strategy: Range-based. This means movies starting with 'A-L' might go to Shard 1, and 'M-Z' to Shard 2.

Replication: Each shard must have a replica factor of 2 (High Availability).thus total of 4 shards 2replicas and 2 original shards

2.Users Collection (Hash-Based Sharding):

Shard Key: _id.

Strategy: Hashed. This ensures an even distribution of users across all shards, regardless of their ID sequence, which is better for write-heavy loads. (don't know if mongo use consistent hashing by default)

Step 2: Understanding The generate movies code

The code is really simple it is 100 lines of code to generate 10000 movies and fill the database to see how sharding and replications work together

The strings are randomly chosen

Step 3: revising mongodb architecture and components

1. Config Server

The Config Server stores metadata about the cluster, not your actual application data

2. Mongos (the router)

The router work with config server hand on hand to process the client request and decide where the query goes

3. Shards server

Stores the data

Step 4: Configuring the database

Database configuration commands will be in [commands.sh](#) for referencing

First we turn on the database

```
x Tue 21 Apr - 09:40 ~/UNI/adistr/MongoShard | origin @ master ↑1 ✓
@drnull sudo systemctl start mongod

Tue 21 Apr - 09:41 ~/UNI/adistr/MongoShard | origin @ master ↑1 ✓
@drnull |
```

We create folders for storing information

```
Tue 21 Apr - 09:41 ~/UNI/adistr/MongoShard | origin @ master ↑1 ✓
@drnull mkdir -p data/config data/s1n1 data/s1n2 data/s2n1 data/s2n2

Tue 21 Apr - 09:43 ~/UNI/adistr/MongoShard | origin @ master ↑1 2+
@drnull ls
└─ commands.sh ── data ── gitlab.txt ── movies-generator-mongodb ── Report

Tue 21 Apr - 09:43 ~/UNI/adistr/MongoShard | origin @ master ↑1 2+
@drnull |
```

Starting the config server

```

mongod --configsvr --replSet cfgrs --port 27016 --dbpath ./data/config --bind_ip
localhost
# Connect via mongosh and run: rs.initiate({_id: "cfgrs", members: [{_id: 0,
host: "localhost:27016"}]})

```

```

conn2":{"msg":"Connection ended","attr":{"remote":"127.0.0.1:40366","isLoadBalanced":false,"uid
d":{"_id":"021f3526-792b-45e2-9d36-12ff5408ad92"},"connectionId":2,"connectionCount
":4}}
{"t":{"sdate":"2026-04-21T09:46:05.303+03:00"},"s":"I", "c":"NETWORK", "id":22944, "ctx":"
conn3","msg":"Connection ended","attr":{"remote":"127.0.0.1:40974","isLoadBalanced":false,"uid
d":{"_id":"2e7919b6-7139-4d83-bd36-7335d7100627"},"connectionId":3,"connectionCount
":3}}
{"t":{"sdate":"2026-04-21T09:46:11.686+03:00"},"s":"I", "c":"NETWORK", "id":6788700, "ctx":"
conn12","msg":"Received first command on ingress connection since session start or auth handsh
ake","attr":{"elapsedMillis":9999}}
{"t":{"sdate":"2026-04-21T09:46:51.409+03:00"},"s":"I", "c":"SH_REFR", "id":4619902, "ctx":"
CatalogCache-1","msg":"Collection has found to be unsharded after refresh","attr":{"namespace
":"config.external_validation_keys","durationMillis":0}}
{"t":{"sdate":"2026-04-21T09:46:51.409+03:00"},"s":"I", "c":"SH_REFR", "id":4619902, "ctx":"
CatalogCache-1","msg":"Collection has found to be unsharded after refresh","attr":{"namespace
":"config.tenantMigrationRecipients","durationMillis":0}}
{"t":{"sdate":"2026-04-21T09:46:51.410+03:00"},"s":"I", "c":"SH_REFR", "id":4619902, "ctx":"
CatalogCache-1","msg":"Collection has found to be unsharded after refresh","attr":{"namespace
":"config.analyzeShardKeySplitPoints","durationMillis":0}}
{"t":{"sdate":"2026-04-21T09:46:51.410+03:00"},"s":"I", "c":"SH_REFR", "id":4619902, "ctx":"
CatalogCache-1","msg":"Collection has found to be unsharded after refresh","attr":{"namespace
":"config.sampledQueriesDiff","durationMillis":0}}
{"t":{"sdate":"2026-04-21T09:46:51.410+03:00"},"s":"I", "c":"SH_REFR", "id":4619902, "ctx":"
CatalogCache-1","msg":"Collection has found to be unsharded after refresh","attr":{"namespace
":"config.sampledQueries","durationMillis":0}}
{"t":{"sdate":"2026-04-21T09:46:51.410+03:00"},"s":"I", "c":"SH_REFR", "id":4619902, "ctx":"
CatalogCache-1","msg":"Collection has found to be unsharded after refresh","attr":{"namespace
":"config.tenantMigrationDonors","durationMillis":0}}
{"t":{"sdate":"2026-04-21T09:47:01.694+03:00"},"s":"I", "c":"WCHKPT", "id":22430, "ctx":"
Checkpoint","msg":"WiredTiger message","attr":{"message":{"ts_sec":1776754021,"ts_usec":6944
52,"thread":"88591:0x701b132576c0"},"session_name":"WT_SESSION.checkpoint","category":"WT_VERB
CHECKPOINT_PROGRESS","category_id":6,"verbose_level":"DEBUG_1","verbose_level_id":1,"msg":"sav
ing checkpoint snapshot min: 415, snapshot max: 415 snapshot count: 0, oldest timestamp: (1776
753961, 1), meta checkpoint timestamp: (1776754020, 1) base write gen: 1"}}}

```

```

Tue 21 Apr - 09:44 ~/UNI/adistr/MongoShard > mongosh --port 27016
Current Mongosh Log ID: 69e71d1a40e6f343044ba88
Connecting to: mongod://127.0.0.1:27016/?directConnection=true&serverSelectionTimeout
ms=2000&appName=mongosh+2.8.2
Using MongoDB: 7.0.31
Using Mongosh: 2.8.2
For mongosh info see: https://www.mongodb.com/docs/mongosh-shell/

-----
The server generated these startup warnings when booting:
2026-04-21T09:44:51.207+03:00: Using the XFS filesystem is strongly recommended with the Wi
redtiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2026-04-21T09:44:51.374+03:00: Access control is not enabled for the database. Read and writ
e access to data and configuration is unrestricted
2026-04-21T09:44:51.374+03:00: Soft limits for open file descriptors too low
-----

test> rs.initiate({_id: "cfgrs", members: [{_id: 0, host: "localhost:27016"}]})
{ ok: 1 }
cfgrs [direct: other] test> exit

```

```

Tue 21 Apr - 09:46 ~/UNI/adistr/MongoShard > mongosh --port 27016
Current Mongosh Log ID: 69e71d1a40e6f343044ba88
Connecting to: mongod://127.0.0.1:27016/?directConnection=true&serverSelectionTimeout
ms=2000&appName=mongosh+2.8.2
Using MongoDB: 7.0.31
Using Mongosh: 2.8.2
For mongosh info see: https://www.mongodb.com/docs/mongosh-shell/

-----
The server generated these startup warnings when booting:
2026-04-21T09:44:51.207+03:00: Using the XFS filesystem is strongly recommended with the Wi
redtiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2026-04-21T09:44:51.374+03:00: Access control is not enabled for the database. Read and writ
e access to data and configuration is unrestricted
2026-04-21T09:44:51.374+03:00: Soft limits for open file descriptors too low
-----

test> rs.initiate({_id: "s1rs", members: [{_id: 0, host: "localhost:27017"}, {_id: 1, host: "localhost:27018"}]})
{ ok: 1 }
s1rs [direct: other] test> exit

```

Starting shard1

```

mongod --shardsvr --replSet s1rs --port 27017 --dbpath ./data/s1n1 --bind_ip
localhost
mongod --shardsvr --replSet s1rs --port 27018 --dbpath ./data/s1n2 --bind_ip
localhost
# Connect to 27017 and run: rs.initiate({_id: "s1rs", members: [{_id: 0, host:
"localhost:27017"}, {_id: 1, host: "localhost:27018"}]})

```

Now we have 3 terminals

```

{"c":{"sdate":"2026-04-21T09:48:01.724+03:00","s":"I","c":"WTCHKPT","id":22430,"ctx":{"Checkpointnr":"msg":"Wiredtiger message" attr":{"message":{"ts_sec":1776754081,"ts_usec":724800,"thread":"88591:0x701b132576c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":6,"verbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 537, snapshot max: 537 snapshot count: 0, oldest timestamp: (1776753961, 1), meta checkpoint timestamp: (1776754080, 1) base write gen: 1}}}}
{"c":{"sdate":"2026-04-21T09:49:01.733+03:00","s":"I","c":"WTCHKPT","id":22430,"ctx":{"Checkpointnr":"msg":"Wiredtiger message" attr":{"message":{"ts_sec":1776754141,"ts_usec":733032,"thread":"88591:0x701b132576c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":6,"verbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 659, snapshot max: 659 snapshot count: 0, oldest timestamp: (1776753961, 1), meta checkpoint timestamp: (1776754140, 1) base write gen: 1}}}}
{"c":{"sdate":"2026-04-21T09:49:51.424+03:00","s":"I","c":"SH_REFR","id":4619902,"ctx":{"CatalogCache-3","msg":"Collection has found to be unsharded after refresh","attr":{"namespace":"config.collections","durationMillis":0}}}}
{"c":{"sdate":"2026-04-21T09:49:51.424+03:00","s":"I","c":"SH_REFR","id":4619902,"ctx":{"CatalogCache-2","msg":"Collection has found to be unsharded after refresh","attr":{"namespace":"config.system.sessions","durationMillis":1}}}}
{"c":{"sdate":"2026-04-21T09:49:51.424+03:00","s":"I","c":"CONTROL","id":28712,"ctx":{"LogicalSessionCacheReap","msg":"Sessions collection is not set up; waiting until next sessions reap interval","attr":{"error":"NamespaceNotSharded: Expected collection config.system.sessions to be sharded"}}}}
{"c":{"sdate":"2026-04-21T09:49:51.425+03:00","s":"I","c":"CONTROL","id":28710,"ctx":{"LogicalSessionCacheRefresh","msg":"Failed to refresh session cache, will try again at the next refresh interval","attr":{"error":"ShardNotFound: Failed to create config.system.sessions: cannot create the collection until there are shards"}}}}
{"c":{"sdate":"2026-04-21T09:50:01.745+03:00","s":"I","c":"WTCHKPT","id":22430,"ctx":{"Checkpointnr":"msg":"Wiredtiger message" attr":{"message":{"ts_sec":1776754201,"ts_usec":74520,"thread":"88591:0x701b132576c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":6,"verbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 781, snapshot max: 781 snapshot count: 0, oldest timestamp: (1776753961, 1), meta checkpoint timestamp: (1776754200, 1) base write gen: 1}}}}
{"c":{"sdate":"2026-04-21T09:51:01.756+03:00","s":"I","c":"WTCHKPT","id":22430,"ctx":{"Checkpointnr":"msg":"Wiredtiger message" attr":{"message":{"ts_sec":1776754261,"ts_usec":756322,"thread":"88591:0x701b132576c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":6,"verbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 903, snapshot max: 903 snapshot count: 0, oldest timestamp: (1776753961, 1), meta checkpoint timestamp: (1776754260, 1) base write gen: 1}}}}
{"c":{"sdate":"2026-04-21T09:52:01.776+03:00","s":"I","c":"WTCHKPT","id":22430,"ctx":{"Checkpointnr":"msg":"Wiredtiger message" attr":{"message":{"ts_sec":1776754321,"ts_usec":769338,"thread":"88591:0x701b132576c0","session_name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":6,"verbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 1025, snapshot max: 1025 snapshot count: 0, oldest timestamp: (1776754020, 1), meta checkpoint timestamp: (1776754320, 1) base write gen: 1}}}}
{"c":{"sdate":"2026-04-21T09:52:31.338+03:00","s":"I","c":"INDEX","id":28345,"ctx":{"TenantMigrationDonorService-0","msg":"Index build: done building","attr":{"buildUUID":"null","collectionUUID":{"uuid":{"$uuid":"0f61732d-7d8e-454d-8bbc-d34f4ae3457d"},"namespace":"config.external_validation_keys","index":{"$index":{"id":"index-46-1496355227043458457","collectionIdent":"collection-45-1496355227043458457","commitTimestamp":{"$timestamp":{"t":1776754351,"l":6}}}}}}}}
{"c":{"sdate":"2026-04-21T09:52:31.338+03:00","s":"I","c":"REPL","id":5123005,"ctx":{"TenantMigrationDonorService-0","msg":"Rebuilding PrimaryOnlyService due to stepUp","attr":{"service":"TenantMigrationDonorService"}}}}
{"c":{"sdate":"2026-04-21T09:52:31.508+03:00","s":"I","c":"M","id":4939300,"ctx":{"monitoring-keys-for-HMAC","msg":"Failed to refresh key cache","attr":{"error":"KeyNotFound: No keys found after refresh","nextWakeUpMillis":5000}}}}
D":{"uuid":{"$uuid":"0f61732d-7d8e-454d-8bbc-d34f4ae3457d"},"namespace":"config.external_validation_keys","index":{"id":"index-48-8609071473615129613","collectionIdent":"collection-47-8609071473615129613","commitTimestamp":{"$timestamp":{"t":1776754351,"l":5}}}}}}
{"c":{"sdate":"2026-04-21T09:52:31.370+03:00","s":"I","c":"REPL","id":7360110,"ctx":{"ReplWriterWorker-0","msg":"Applying DDL command oplog entry","attr":{"oplogEntry":{"oplogEntry":{"op":"c","ns":"config.scm","ui":{"$uuid":"0f61732d-7d8e-454d-8bbc-d34f4ae3457d"},"createIndexes":{"external_validation_keys","v":2,"key":{"ttlExpiresAt":{"$timestamp":{"t":1776754351,"l":5}}},"expireAfterSeconds":0,"ttl":{"$timestamp":{"t":1776754351,"l":6}}},"v":2,"wall":{"$date":"2026-04-21T09:52:31.352Z"}}},"oplogApplicationMode":"Secondary"}}}}
{"c":{"sdate":"2026-04-21T09:52:31.373+03:00","s":"I","c":"INDEX","id":28345,"ctx":{"ReplWriterWorker-0","msg":"Index build: done building","attr":{"buildUUID":"null","collectionUUID":{"uuid":{"$uuid":"0f61732d-7d8e-454d-8bbc-d34f4ae3457d"},"namespace":"config.external_validation_keys","index":{"$index":{"id":"index-49-8609071473615129613","collectionIdent":"collection-47-8609071473615129613","commitTimestamp":{"$timestamp":{"t":1776754351,"l":6}}}}}}}}
{"c":{"sdate":"2026-04-21T09:52:31.630+03:00","s":"I","c":"STORAGE","id":22260,"ctx":{"TimestampMonitor","msg":"Removing drop-pending idents with drop timestamps before timestamp","attr":{"timestamp":{"$timestamp":{"t":1776754340,"l":1}}}}}}
{"c":{"sdate":"2026-04-21T09:52:31.630+03:00","s":"I","c":"STORAGE","id":22237,"ctx":{"TimestampMonitor","msg":"Completing drop for ident","attr":{"ident":{"collection-15-8609071473615129613","dropTimestamp":{"$timestamp":{"t":0,"l":0}}}}}}}}
{"c":{"sdate":"2026-04-21T09:52:31.637+03:00","s":"I","c":"STORAGE","id":6776600,"ctx":{"TimestampMonitor","msg":"The ident was successfully dropped","attr":{"ident":{"collection-15-8609071473615129613","dropTimestamp":{"$timestamp":{"t":0,"l":0}}}}}}

```

The one on the left is the config server

And the two on the right are shard1

Moving on to shard2

```

mongod --shardsvr --replSet s2rs --port 27019 --dbpath ./data/s2n1 --bind_ip
localhost
mongod --shardsvr --replSet s2rs --port 27020 --dbpath ./data/s2n2 --bind_ip
localhost

# Connect to 27019 and run: rs.initiate({_id: "s2rs", members: [{_id: 0, host:
"localhost:27019"}, {_id: 1, host: "localhost:27020"}]})

```

Two more terminals for shard2

```

{"t":{"$date":"2026-04-21T09:54:56.151+03:00"},"s":"I", "c":"INDEX", "id":28345, "ctx":"
TenantMigrationDonorService-0", "msg":"Index build: done building", "attr":{"buildUUID":null,"co
llectionUUID":{"uid":{"$uid":"d66a9d9-2227-4b87-987a-7dafef21b212"}}, "namespace":"config.ex
ternal_validation_keys", "index":{"ExternalKeysTLIndex", "ident":{"index-47-1133282580286577686",
"collectionid":"collection-45-1133282580286577686", "commitTimestamp":{"t":{"$timestamp":"177
675496","i":{"$inc":6}}}}}}}}
{"t":{"$date":"2026-04-21T09:54:56.151+03:00"},"s":"I", "c":"REPL", "id":5123005, "ctx":"
TenantMigrationDonorService-0", "msg":"Rebuilding PrimaryOnlyService due to stepUp", "attr":{"se
rvice":{"TenantMigrationDonorService"}}}
{"t":{"$date":"2026-04-21T09:54:59.240+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":4800}}
{"t":{"$date":"2026-04-21T09:55:04.046+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":5800}}
{"t":{"$date":"2026-04-21T09:55:08.052+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":5200}}
{"t":{"$date":"2026-04-21T09:55:13.567+03:00"},"s":"I", "c":"CONNPOL", "id":22576, "ctx":"
Replicetwork", "msg":"Connecting", "attr":{"hostAndPort":"localhost:27020"}}
{"t":{"$date":"2026-04-21T09:55:14.257+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":5400}}
{"t":{"$date":"2026-04-21T09:55:19.663+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":5800}}
{"t":{"$date":"2026-04-21T09:55:31.075+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":5600}}
{"t":{"$date":"2026-04-21T09:55:25.268+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":6200}}
{"t":{"$date":"2026-04-21T09:55:37.082+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":6800}}
{"t":{"$date":"2026-04-21T09:55:43.288+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":6400}}
{"t":{"$date":"2026-04-21T09:55:45.246+03:00"},"s":"I", "c":"WTCHKPT", "id":22430, "ctx":"
Checkpoint", "msg":"Wiredtiger message", "attr":{"message":{"ts_sec":1776754545, "ts_usec":2459
74, "thread":{"111920:0x74bc5b5b6c0", "session_name":"WT_SESSION_checkpoint", "category":"WT_VERB
CHECKPOINT_PROGRESS", "category_id":16, "verbose_level":"DEBUG", "verbose_level_id":1, "msg":"ea
ving checkpoint snapshot min: 201, snapshot max: 201 snapshot count: 0, oldest timestamp: (177
675485, 1), meta checkpoint timestamp: (1776754535, 1) base write gen: 1}}}}
{"t":{"$date":"2026-04-21T09:55:49.696+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":6600}}
{"t":{"$date":"2026-04-21T09:54:57.266+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":4200}}
{"t":{"$date":"2026-04-21T09:55:01.471+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":4400}}
{"t":{"$date":"2026-04-21T09:55:05.876+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":4600}}
{"t":{"$date":"2026-04-21T09:55:10.492+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":4800}}
{"t":{"$date":"2026-04-21T09:55:13.567+03:00"},"s":"I", "c":"NETWORK", "id":22943, "ctx":"
listener", "msg":"Connection accepted", "attr":{"remote":{"127.0.0.1:41002", "isLoadBalanced":fals
e, "uid":{"uid":{"$uid":"1059d3a6-260a-41a4-877f-9e83be777f87"}}, "connectionId":12, "connecti
onCount":2}}}}
{"t":{"$date":"2026-04-21T09:55:13.567+03:00"},"s":"I", "c":"NETWORK", "id":51800, "ctx":"
conn12", "msg":"client metadata", "attr":{"remote":{"127.0.0.1:41002", "client":"conn12", "negotiat
edCompressors":["snappy", "zstd", "zlib"], "doc":{"driver":{"name":"NetworkInterfaceTL-ReplNetwor
k", "version":"7.0.31"}, "os":{"type":"Linux", "name":"Ubuntu", "architecture":"x86_64", "version":
"24.04"}}}}}}
{"t":{"$date":"2026-04-21T09:55:15.288+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":5800}}
{"t":{"$date":"2026-04-21T09:55:15.567+03:00"},"s":"I", "c":"NETWORK", "id":6788700, "ctx":"
conn12", "msg":"Received first command on ingress connection since session start or auth handsh
ake", "attr":{"elapsedMillis":1939}}
{"t":{"$date":"2026-04-21T09:55:20.294+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":5200}}
{"t":{"$date":"2026-04-21T09:55:25.493+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":5400}}
{"t":{"$date":"2026-04-21T09:55:30.905+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":5600}}
{"t":{"$date":"2026-04-21T09:55:36.510+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":5800}}
{"t":{"$date":"2026-04-21T09:55:42.316+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":6000}}
{"t":{"$date":"2026-04-21T09:55:48.322+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":6200}}
{"t":{"$date":"2026-04-21T09:55:54.528+03:00"},"s":"I", "c":"-", "id":4933300, "ctx":"
monitoring-keys-for-HMAC", "msg":"Failed to refresh key cache", "attr":{"error":"KeyNotFound: No
keys found after refresh", "nextWakeUpMillis":6400}}

```

Now configuring the router


```
mongos --configdb cfgrs/localhost:27016 --port 27021 --bind_ip localhost
```

We then connect to the router mongosh –port 27021 and enter the following configurations

```
// Add Shards to the cluster
sh.addShard("s1rs/localhost:27017");
sh.addShard("s2rs/localhost:27019");

// Enable sharding for the database
sh.enableSharding("videodb");

// Shard the Movies collection (Range-based on
// title)
// Note: You must create an index on the shard key
// first
use videodb;
db.movies.createIndex({ title: 1 });
sh.shardCollection("videodb.movies", { title: 1 });

// Shard the Users collection (Hashed on _id)
sh.shardCollection("videodb.users", { _id: "hashed"
```

```
});
```

Step 5 writing the java spring boot app :

No need to explain this step we already did this in our previous homework

We just connect User and Movie model with actual database collections

And we run our stress test using something similar to the provided code

“<https://git.hiast.edu.sy/mohamadbashar.disoki/movies-generator-mongodb>” but not exactly the same

Note: in application.properties we connect to mongos (the router)

Step 6 running the code :

We run the code using the following command

```
mvn spring-boot:run
```

```

:: Spring Boot :: (v3.2.2)

2026-04-21T09:35:41.857+02:00 INFO 207500 --- [main] sy.edu.hiast.Application : Starting Application using Java 21.0.8 with PID 207500 (/home/drmull/UNI/adistr/M
ngoShard/movie-shard-app/target/classes started by drnull in /home/drmull/UNI/adistr/MongoShard/movie-shard-app)
2026-04-21T09:35:41.859+02:00 INFO 207500 --- [main] sy.edu.hiast.Application : No active profile set, falling back to 1 default profile: "default"
2026-04-21T09:35:42.065+02:00 INFO 207500 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data MongoDB repositories in DEFAULT mode.
2026-04-21T09:35:42.081+02:00 INFO 207500 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 13 ms. Found 0 MongoDB repository int
erfaces.
2026-04-21T09:35:42.277+02:00 INFO 207500 --- [main] org.mongodb.driver.client : MongoClient with metadata {"driver": {"name": "mongo-java-driver|syncspring-boot
", "version": "4.11.1"}, "os": {"type": "Linux", "name": "Linux", "architecture": "amd64", "version": "6.14.0-35-generic"}, "platform": "Java/Ubuntu/21.0.8+9-Ubuntu-0ubuntu124.04.1"} create
d with settings MongoClientSettings(readPreference=primary, writeConcern=WriteConcern(w=null, wTimeout=null ms, journal=null), retryWrites=true, retryReads=true, readConcern=ReadConcern(lev
el=null), credential=null, transportSettings=null, streamFactory=null, commandListeners=[], codecRegistry=ProvidersCodecRegistry(codecProviders=[ValueCodecProvider{}, BsonValueCodecP
rovider{}, DBRefCodecProvider{}, DBObjectCodecProvider{}, DocumentCodecProvider{}, CollectionCodecProvider{}, IterableCodecProvider{}, MapCodecProvider{}, GeoJsonCodecProvider{}, GridFSFile
CodecProvider{}, Jsr310CodecProvider{}, JsonObjectCodecProvider{}, BsonObjectCodecProvider{}, EnumCodecProvider{}, com.mongodb.client.model.mql.ExpressionCodecProvider@15f87a32, com.mongodb.Jep39
RecordCodecProvider@257ccffc, com.mongodb.KotlinCodecProvider@42e22a53]), loggerSettings=LoggerSettings(maxDocumentLength=1000), clusterSettings=(hosts=[localhost:27021], srvServiceName=mo
ngodb, mode=SINGLE, requiredClusterType=UNKNOWN, requiredReplicaSetName=null, serverSelector=null, clusterListeners=[]), serverSelectionTimeout=30000 ms, localThreshold=15 ms}, soc
ketSettings=SocketSettings(connectTimeoutMS=10000, readTimeoutMS=0, receiveBufferSize=0, proxySettings=ProxySettings(host=null, port=null, username=null, password=null)), heartbeatSocketSet
tings=SocketSettings(connectTimeoutMS=10000, readTimeoutMS=10000, receiveBufferSize=0, proxySettings=ProxySettings(host=null, port=null, username=null, password=null)), connectionPoolSettin
gs=ConnectionPoolSettings(maxSize=100, minSize=0, maxWaitTimeMS=120000, maxConnectionIdleTimeMS=0, maxConnectionLifeTimeMS=0, maintenanceInitialDelayMS=0, maintenanceFrequencyMS=60000, con
nectionPoolListeners=[], maxConnecting=2), serverSettings=ServerSettings(heartbeatFrequencyMS=10000, minHeartbeatFrequencyMS=500, serverListeners=[], serverMonitorListeners=[]), sslSetti
ngs=SslSettings(enabled=false, invalidHostNameAllowed=false, context=null), applicationName=null, compressorList=[], uuidRepresentation=JAVA_LEGACY, serverApi=null, autoEncryptionSettings
=null, dnsClient=null, inetAddressResolver=null, contextProvider=null)
2026-04-21T09:35:42.294+02:00 INFO 207500 --- [localhost:27021] org.mongodb.driver.cluster : Monitor thread successfully connected to server with description ServerDescriptio
n(address=localhost:27021, type=SHARD_ROUTER, state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=21, maxDocumentSize=16777216, logicalSessionTimeoutMinutes=30, roundTripTimeNanos=10
389678)
2026-04-21T09:35:42.463+02:00 INFO 207500 --- [main] sy.edu.hiast.Application : Started Application in 0.806 seconds (process running for 0.974)
Starting data generation...
Generation complete!
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 9.835 s
[INFO] Finished at: 2026-04-21T09:35:50+02:00
[INFO] -----

Tue 21 Apr - 10:35 ~/UNI/adistr/MongoShard/movie-shard-app 207500@kali: T1

```

Step 7 verifying the database :

We first connect to the router

```

Tue 21 Apr - 10:35 ~/UNI/adistr/MongoShard/movie-shard-app 207500@kali: T1
drnull@kali:~$ mongosh --port 27021
Current Mongosh Log ID: 69e729ba8f6904620d44ba88
Connecting to: mongod://127.0.0.1:27021/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.8.2
Using MongoDB: 7.0.31
Using Mongosh: 2.8.2

For mongosh info see: https://www.mongodb.com/docs/mongosh-shell/

-----
The server generated these startup warnings when booting
2026-04-21T09:56:36.862+03:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----

[direct: mongos] test>

```

```

Tue 21 Apr - 10:35 ~/UNI/adistr/MongoShard/movie-shard-app > mongosh --port 27021
Current Mongosh Log ID: 69e729ba8f6904620d44ba88
Connecting to: mongod://127.0.0.1:27021/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.8.2
Using MongoDB: 7.0.31
Using Mongosh: 2.8.2

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

-----
The server generated these startup warnings when booting
2026-04-21T09:56:36.862+03:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----

[direct: mongos] test> use videodb;
switched to db videodb
[direct: mongos] videodb> db.movies.getShardDistribution();
Shard s2rs at s2rs/localhost:27019,localhost:27020
{
  data: '1.36MiB',
  docs: 10000,
  chunks: 1,
  'estimated data per chunk': '1.36MiB',
  'estimated docs per chunk': 10000
}
-----
Totals
{
  data: '1.36MiB',
  docs: 10000,
  chunks: 1,
  'Shard s2rs': [
    '100 % data',
    '100 % docs in cluster',
    '1438 avg obj size on shard'
  ]
}
[direct: mongos] videodb>

```

The data was so small below the chunk limit

So it wasn't distributed to fix this we should make the chunk size smaller 1mg


```
[direct: mongos] videodb> exit

Tue 21 Apr - 10:44 ~/UNI/adistr/MongoShard/movie-shard-app
$ mongosh --port 27021
Current Mongosh Log ID: 69e72ac97dd780604d44ba88
Connecting to: mongodb://127.0.0.1:27021/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.8.2
Using MongoDB: 7.0.31
Using Mongosh: 2.8.2

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

-----
The server generated these startup warnings when booting
2026-04-21T09:56:36.862+03:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----

[direct: mongos] test> use config;
switched to db config
[direct: mongos] config> db.settings.updateOne(
  {
    _id: "chunksize",
    $set: { value: 1 },
    { upsert: true }
  }
);
{
  acknowledged: true,
  insertedId: 'chunksize',
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1
}
[direct: mongos] config>
```

Now we wait for it to rebalance around 40 sec just to make sure

After some waiting the output of the sh.status()

```
sh.status();
shardingVersion
{ _id: 1, clusterId:
ObjectId('69e71d29eb3b7f7564584362') }
---
shards
[
  {
    _id: 's1rs',
```

```
    host: 's1rs/localhost:27017,localhost:27018',
    state: 1,
    topologyTime: Timestamp({ t: 1776754743, i: 2 })
  },
  {
    _id: 's2rs',
    host: 's2rs/localhost:27019,localhost:27020',
    state: 1,
    topologyTime: Timestamp({ t: 1776754766, i: 2 })
  }
]
---
active mongoses
[ { '7.0.31': 1 } ]
---
autosplit
{ 'Currently enabled': 'yes' }
---
balancer
{
  'Currently enabled': 'yes',
  'Currently running': 'no',
  'Failed balancer rounds in last 5 attempts': 0,
```

```
'Migration Results for the last 24 hours': 'No
recent migrations'
}
---
shardedDataDistribution
[
  {
    ns: 'videodb.movies',
    shards: [
      {
        shardName: 's2rs',
        numOrphanedDocs: 0,
        numOwnedDocuments: 10000,
        ownedSizeBytes: 1430000,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'videodb.users',
    shards: [
      {
        shardName: 's1rs',
```

```
    numOrphanedDocs: 0,  
    numOwnedDocuments: 2525,  
    ownedSizeBytes: 409050,  
    orphanedSizeBytes: 0  
  },  
  {  
    shardName: 's2rs',  
    numOrphanedDocs: 0,  
    numOwnedDocuments: 2475,  
    ownedSizeBytes: 400950,  
    orphanedSizeBytes: 0  
  }  
]  
},  
{  
  ns: 'config.system.sessions',  
  shards: [  
    {  
      shardName: 's1rs',  
      numOrphanedDocs: 0,  
      numOwnedDocuments: 12,  
      ownedSizeBytes: 1188,  
      orphanedSizeBytes: 0
```

```

    }
  ]
}
]
---
databases
[
  {
    database: { _id: 'config', primary: 'config',
partitioned: true },
    collections: {
      'config.system.sessions': {
        shardKey: { _id: 1 },
        unique: false,
        balancing: true,
        allowMigrations: true,
        chunkMetadata: [ { shard: 's1rs', nChunks: 1
} ],
        chunks: [
          { min: { _id: MinKey() }, max: { _id:
MaxKey() }, 'on shard': 's1rs', 'last modified':
Timestamp({ t: 1, i: 0 }) }
        ],

```

```
      tags: []
    }
  },
  {
    database: {
      _id: 'videodb',
      primary: 's2rs',
      partitioned: false,
      version: {
        uuid:
UUID('472c5ca2-2844-48d0-945e-67bb0f6adde6'),
        timestamp: Timestamp({ t: 1776754774, i: 1
      }),
      lastMod: 1
    }
  },
  collections: {
    'videodb.movies': {
      shardKey: { title: 1 },
      unique: false,
      balancing: true,
      allowMigrations: true,
```

```

    chunkMetadata: [ { shard: 's2rs', nChunks:
1 } ],
    chunks: [
      { min: { title: MinKey() }, max: { title:
MaxKey() }, 'on shard': 's2rs', 'last modified':
Timestamp({ t: 1, i: 0 }) }
    ],
    tags: []
  },
  'videodb.users': {
    shardKey: { _id: 'hashed' },
    unique: false,
    balancing: true,
    allowMigrations: true,
    chunkMetadata: [
      { shard: 's1rs', nChunks: 1 },
      { shard: 's2rs', nChunks: 1 }
    ],
    chunks: [
      { min: { _id: MinKey() }, max: { _id:
Long('0') }, 'on shard': 's2rs', 'last modified':
Timestamp({ t: 1, i: 5 }) },
      { min: { _id: Long('0') }, max: { _id:

```

```
MaxKey() }, 'on shard': 's1rs', 'last modified':  
Timestamp({ t: 1, i: 4 }) }  
  ],  
  tags: []  
}  
}  
}  
]
```

As we can see the user collection is balanced 2500 record on each shard

But the problem is still present in the movie shard it is still one chunk on one shard

For that i will try to forcibly splitting it

First we use the db

```
[direct: mongos] test> use videodb;  
switched to db videodb
```

Then we split on letter M


```

switched to db videodb
[direct: mongos] videodb> sh.splitFind("videodb.movies", { title: "M" });
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1776758038, i: 7 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1776758038, i: 7 })
}

```

Then we manually move data instead of waiting for the balancer

```

operationTime: Timestamp({ t: 1776758054, i: 5044 })
}
[direct: mongos] videodb> sh.moveChunk("videodb.movies", { title: "A" }, "s1rs");
{
  millis: 209,
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1776758054, i: 5044 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1776758054, i: 5044 })
}

```

```
operationTime: Timestamp({ t: 147030034, l: 3044 })
}
[direct: mongos] videodb> db.movies.getShardDistribution();
Shard s2rs at s2rs/localhost:27019,localhost:27020
{
  data: '698KiB',
  docs: 10000,
  chunks: 1,
  'estimated data per chunk': '698KiB',
  'estimated docs per chunk': 10000
}
---
Shard s1rs at s1rs/localhost:27017,localhost:27018
{
  data: '698KiB',
  docs: 5000,
  chunks: 1,
  'estimated data per chunk': '698KiB',
  'estimated docs per chunk': 5000
}
---
Totals
{
  data: '1.36MiB',
  docs: 15000,
  chunks: 2,
  'Shard s2rs': [
    '50 % data',
    '66.66 % docs in cluster',
    '143B avg obj size on shard'
  ],
  'Shard s1rs': [
    '50 % data',
    '33.33 % docs in cluster',
    '143B avg obj size on shard'
  ]
}
[direct: mongos] videodb> █
```

It worked but why is there 15000 ?

It seems that i ran the java code twice by mistake so we have 20k instead of 10k document

And since it is range base we don't expect uniform distribution

Thanks For Reading

