Sharding in MongoDB

Efficiently Scaling and Managing Large Datasets

Chapter 08

Instructor: Houssein Dhayne
houssein.dhayne@net.usj.edu.lb

Table of content

Introduction to MongoDB

- MongoDB, the most popular NoSQL database, is an open-source document-oriented database.
- The term 'NoSQL' means 'non-relational'.
  - It means that MongoDB isn't based on the table-like relational database structure but provides an altogether different mechanism for storage and retrieval of data.
- This format of storage is called BSON (similar to JSON format).

MongoDB Document Structure

- Example of a Simple MongoDB Document

```
{
  title: 'MongoDB',
  by: 'Harshit Gupta',
  url: 'https://www.MongoDB.org',
  type: 'NoSQL'
}
```

- BSON Format: Similar to JSON

Inserting Documents in MongoDB

- Inserting documents is achieved through the insert method, defining key-value pairs.
- Connect to your MongoDB server
  - use your_database_name;
- Create a collection named "Example"
  - db.createCollection("Example");
- Insert a sample document into the "Example" collection
  - db.Example.insert({
  - key1: "value1",
  - key2: "value2",
  - key3: "value3"
  - });

Retrieving Documents in MongoDB

- Retrieving documents utilizes the find method, allowing for tailored queries.
- Find all documents in the collection
  - db.Example.find();
- Find documents where key1 equals "value1"
  - db.Example.find({ key1: "value1" });
- Suppose we have a collection named "Products":
  - Find documents where the price is greater than $50
    - db.Products.find({ price: { $gt: 50 } });
  - Find documents where the category is "Electronics"
    - db.Products.find({ category: "Electronics" });

Limitations of SQL Databases

- Relational Database Management System(RDBMS) is not the correct choice when it comes to handling big data by the virtue of their design since they are not horizontally scalable.
- If the database runs on a single server, then it will reach a scaling limit.
- NoSQL databases are more scalable and provide superior performance.
- MongoDB is such a NoSQL database that scales by adding more and more servers and increases productivity with its flexible document model.

RDBMS vs MongoDB

- RDBMS has a typical schema design; MongoDB is document-oriented with no concept of schema
- Complex transactions not supported in MongoDB due to the absence of complex join operations
- MongoDB allows flexible and scalable document structures
- MongoDB is faster than RDBMS due to efficient indexing and storage techniques

Common Terms in RDBMS and MongoDB

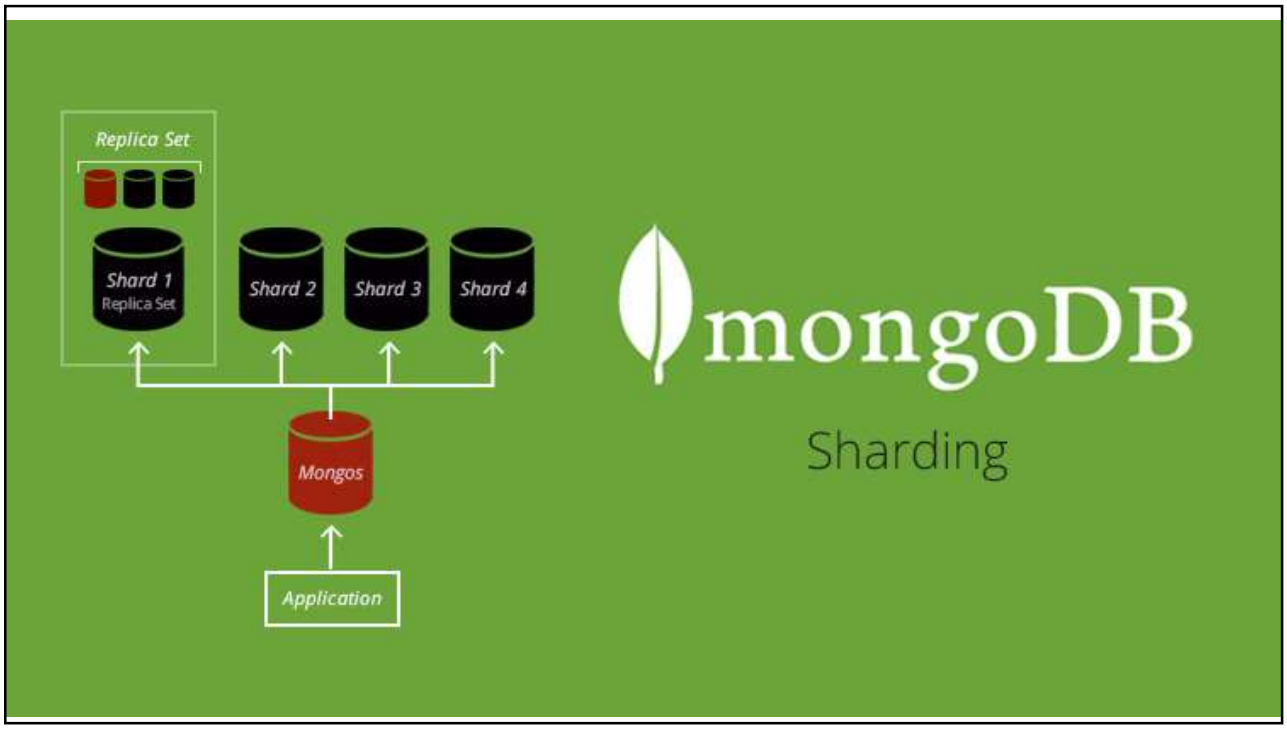
- Table in RDBMS = Collection in MongoDB
- Row in RDBMS = Document in MongoDB
- Column in RDBMS = Field in MongoDB
- Default '_id' in MongoDB similar to Primary key in RDBMS

Features of MongoDB

- Document-oriented: Minimal number of documents, not broken into multiple relational structures
- Indexing: Essential for efficient searching, MongoDB uses indexing to process data quickly
- Scalability: MongoDB scales horizontally using sharding, distributing data across servers
- Replication and High Availability: Multiple copies of data on different servers for data availability and protection
- Aggregation: Operations similar to GROUPBY in SQL, includes sum, avg, min, max, etc.

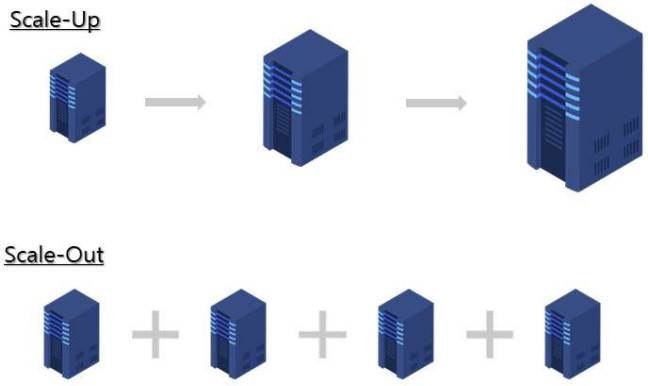
Use Cases for MongoDB

- Big Data: Ideal for storing large amounts of data with built-in solutions for partitioning and sharding
- Unstable Schema: Schema-less nature allows easy addition of new fields without affecting old documents
- Distributed Data: Multiple copies stored across servers ensure instant and safe data recovery



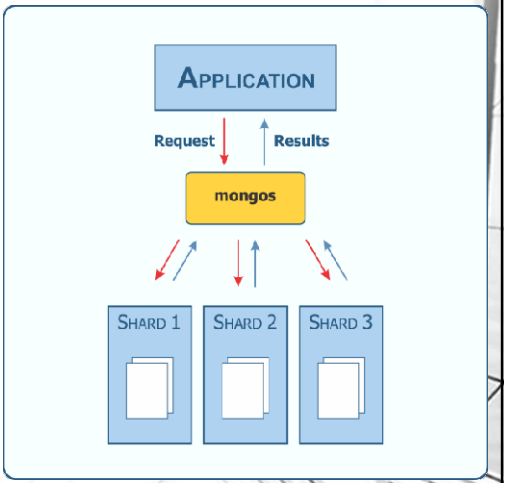
Introduction to Sharding in MongoDB

- When the write workload surpasses a single MongoDB instance's capacity, we turn to sharded clusters.
- Sharding allows us to distribute data across multiple nodes, addressing the limitations of a single cluster.
- Before diving into sharding, it's crucial to optimize your application workload and server configurations



Sharding Fundamentals

- In a sharded database cluster, collections are partitioned across multiple instances, each referred to as a "shard."
- The partitioning is based on a designated "shard key" value, crucial for determining the placement of documents.
- While replica sets focus on high availability, sharding aims at achieving greater scalability.
- Sharding becomes necessary when your workload, especially write operations, exceeds the capacity of a single server.



Scaling and Sharding

- Sharding is an architectural pattern designed to support massive workloads in the world's largest websites.
- As your application load grows, scaling up a single server becomes insufficient.
- Sharding enables "scaling out" by adding more primary nodes and distributing the workload across them.
- Notable early adopters of large-scale sharding include Facebook and Twitter, though the process with MySQL was challenging.
- MongoDB, in contrast, integrates sharding seamlessly into the core database, offering ease of configuration and management.

Sharding Concepts

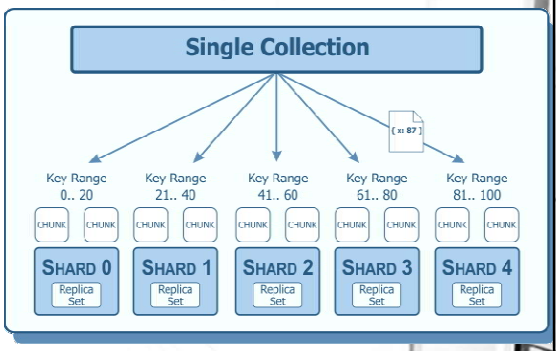
- Sharding involves significant performance opportunities and challenges.
- The shard key, an attribute determining document placement, must have high cardinality for even distribution.
- Documents are organized into "chunks," allocated to specific shards, preventing the need for moving individual documents across shards.
- Two main sharding strategies: Range sharding groups contiguous keys in the same chunk, while Hash sharding distributes keys based on a hash function.
- The balancer in MongoDB ensures data and workload balance across shards by periodically moving data between them.

To Shard or Not to Shard?

- Sharding, the most sophisticated MongoDB configuration, is used by major websites for performance.
- However, it adds complexity and processing overhead, making individual operations slightly slower in many cases.
- Sharding should only be considered as a last resort after optimizing workload, server, and replica set configurations.
- Sharding projects should commence only after exhausting other tuning measures and scale-up options.

Shard Key Selection

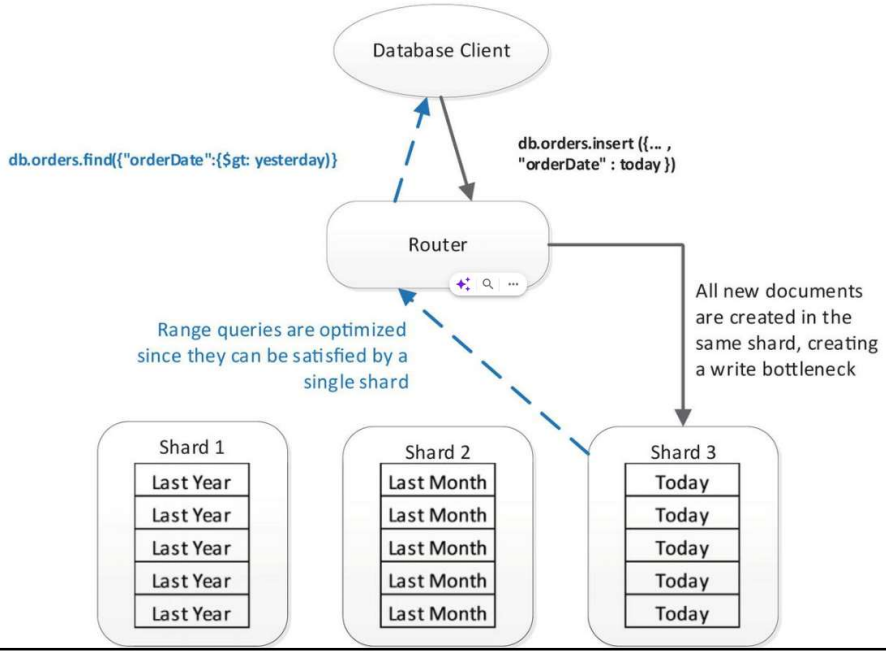
- Sharding operates at the collection level, and not all collections need to be sharded.
- Collections should be sharded if the aggregate IO write demand exceeds the capacity of a single primary.
- Choosing the shard key is critical, considering high cardinality, even distribution of values, and frequent inclusion in queries.
- The shard key should be non-monotonically increasing to avoid creating hot spots and negatively impacting performance.



Range- vs. Hash-Based Sharding

- Distribution of data across shards can be range-based or hash-based.
- Range-based partitioning allocates specific ranges of shard key values to each shard, ensuring even distribution.
- Hash-based sharding uses a hash function on the shard key, resulting in even distribution but challenges with range queries.

Orders collection shared by orderDate

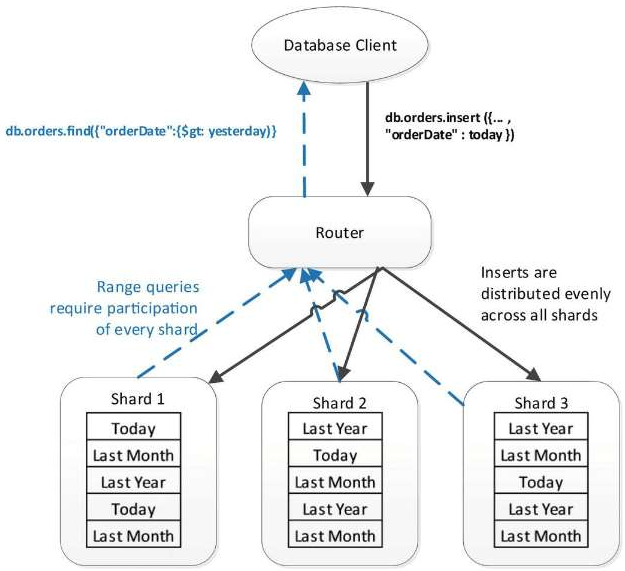


how MongoDB benefits from Range Sharding

Suppose you have a MongoDB cluster that stores order data for an e-commerce platform. Each order document has a unique order ID and a timestamp indicating when the order was placed. You've chosen to range shard the collection based on the order timestamp

- Range sharding is beneficial when queries involve ranges of values for the shard key. For example, queries that retrieve orders within a specific time period.
- The contiguous grouping of timestamp ranges into chunks ensures that querying within a specific time range involves scanning fewer chunks, improving query performance.

Order collection sharded by Hashed orderDate



how MongoDB benefits from Hash Sharding.

Suppose you have a MongoDB cluster that stores user data for a social media platform. The user documents have a unique user ID, and you've chosen to hash shard the collection based on the user ID.

- **Even Distribution:**
  - Hash sharding distributes user documents across shards based on the hash of their user IDs.
  - This even distribution ensures that the workload is spread evenly across all shards, preventing any single shard from becoming a hotspot due to a specific range of user IDs being more frequently accessed.
- **Random Access:**
  - Social media applications often have queries that involve looking up specific users based on their user IDs.
  - With hash sharding, even if there is a high volume of user ID-based queries, the hash function ensures that users are distributed randomly across shards. As a result, these queries can be executed in parallel across multiple shards, improving performance.

Zone Sharding

- Zone sharding allows fine-tuning document distribution across shards.
- It associates a shard with a zone and determines where specific documents will reside based on key ranges.
- Zones can be used to reduce network latency by placing data close to the applications that need it.
- Another use is to distribute "hot" data on powerful hardware and archive older data on cost-effective, slower storage.
- The administration of zones involves allocating shards to zones and assigning shard key ranges to each zone.

Sharding Practice - Setting Up MongoDB Instances

- In this hands-on practice, we'll guide you through setting up a sharded MongoDB environment.
- Step 1 involves downloading and installing MongoDB Community Edition from the official website.
- Following the installation wizard ensures a successful setup for the subsequent sharding configuration.
- Step 2 instructs you to create separate directories for each MongoDB instance: shard1, shard2, shard3, and master.
- These directories will be essential for organizing and managing individual MongoDB instances.

Configuring MongoDB Instances - Step 3

- After creating directories, Step 3 involves crafting configuration files for each MongoDB instance.
- Specific configurations are detailed for the master instance (mongod_config_master.conf) and shards (mongod_config_shard1.conf, mongod_config_shard2.conf, mongod_config_shard3.conf).
- Configuration files dictate system logs, storage paths, network settings, replication, and sharding roles.
- These configurations set the foundation for the subsequent MongoDB instances' behavior.

Starting MongoDB Instances - Step 4

- Step 4 focuses on starting MongoDB instances using command prompt windows and the provided configuration files.
  - mongod –config E:\courses\shards\master\mongod_config_master.conf –replSet configReplSet
- Commands for shard1, shard2, and shard3 follow the same pattern, ensuring that each instance is configured with its designated role and settings.
  - mongod –config E:\courses\shards\shard1\mongod_config_shard.conf –replSet shard1
- Starting MongoDB instances is a critical step in preparing the environment for sharding.

Replica Set Initialization and Configuration - Step 5

- In Step 5, we connect to MongoDB instances and initialize replica sets using the MongoDB shell.
  - mongosh –port 27010 open a connection,
- initializes the replica set:
  - rs.initiate()
- This process is repeated for shard1, shard2, and shard3, ensuring each replica set is correctly configured.
  - rs.initiate({_id: "shard1", members: [{_id: 0, host: "localhost:27021"}]});
- A correctly configured replica set is a prerequisite for a successful sharding setup.

MongoDB Router Configuration - Step 6

- Moving on to Step 6, we start the MongoDB router (mongos) to facilitate communication between shards and applications.
- The mongos –configdb configReplSet/localhost:27010 –port 27011 command initializes the router.
- Connecting to the router via mongosh –port 27011, we prepare to add shards to the sharded environment.
- The subsequent commands demonstrate adding shard1, shard2, and shard3 to the configuration.
  - sh.addShard("shard1/localhost:27021");

Choosing a Sharding Key - Step 1

- Now that our MongoDB environment is set up, we move to the sharding-specific steps.
- Step 1 involves selecting a sharding key. For our "Product" collection, let's choose the "date" field.
- The sharding key plays a crucial role in how data will be distributed across shards.
- In our case, the "date" field will serve as the basis for sharding, allowing for efficient range-based sharding.

Creating Index on Sharding Key - Step 2

- Following the key selection, Step 2 requires creating an index on the chosen sharding key ("date").
- The command db.Product.createIndex({ "date": 1 }) ensures an index is established on the "date" field.
- This index is essential for supporting efficient sharding operations and queries based on the chosen key.
- Index creation is a prerequisite for enabling sharding on the selected database.

Enabling Sharding for Database - Step 3

- With the index in place, Step 3 involves enabling sharding for the target database ("mytest").
- The command sh.enableSharding("mytest") activates sharding on the specified database.
- This step prepares the database for the subsequent sharding of specific collections.

Sharding the Collection Based on Date Ranges - Step 4

- Step 4 focuses on sharding the "Product" collection based on date ranges.
- The command sh.shardCollection("mytest.Product", { "date": 1 }) initiates the sharding process.
- This step ensures that the data within the "Product" collection is distributed across shards based on the selected sharding key ("date").
- The efficiency of range queries will benefit from this sharding strategy.

Tagging and Zone Sharding - Step 5

- Step 5 introduces tagging and zone sharding, offering more granular control over data distribution.
- Shards are tagged with identifiers (e.g., "tag1", "tag2", "tag3").
- Tag ranges are then assigned to each shard based on date ranges.
- This practice optimizes data distribution, allowing for considerations like geographic proximity or hardware specifications.
- Zone sharding enhances performance and flexibility in managing sharded data.

Sample Data Insertion - Step 6

- As a final step, let's insert sample documents into the sharded "Product" collection, considering the defined date ranges for each shard.
- Sample documents showcase the process of associating data with specific shards based on the sharding key ("date").
- The provided commands illustrate how to insert data into each shard corresponding to the predefined date ranges.
- The success of zone sharding becomes evident as data is intelligently distributed across shards.

Conclusion

- In conclusion, this session delved into the practical aspects of sharding in MongoDB, a crucial strategy for handling large datasets and demanding workloads.
- We navigated through the setup of a sharded MongoDB environment, emphasizing key steps such as choosing a sharding key, creating an index, and enabling sharding.
- The significance of thoughtful planning and configuration in achieving an efficient sharded system was underscored throughout the session.
- Sharding, when implemented judiciously, enhances scalability and performance, distributing data intelligently across multiple shards.
- As you embark on your MongoDB journey, remember to consider factors like query patterns, workload characteristics, and ongoing maintenance for a well-optimized sharded environment.
- MongoDB's sharding capabilities empower you to scale horizontally, ensuring your database seamlessly evolves with the growth of your applications and datasets.