g·able

MongoDB 3.0

Sharding

Topics

- Shard Concepts and Architecture
- Data Partitioning
- Performance
- Labs

Sharding Introduction

- Database systems with large data sets and high throughput applications can challenge the capacity of a single server.

- High query rates can exhaust the CPU capacity of the server. Larger data sets exceed the storage capacity of a single machine. Finally, working set sizes larger than the system's RAM stress the I/O capacity of disk drives.

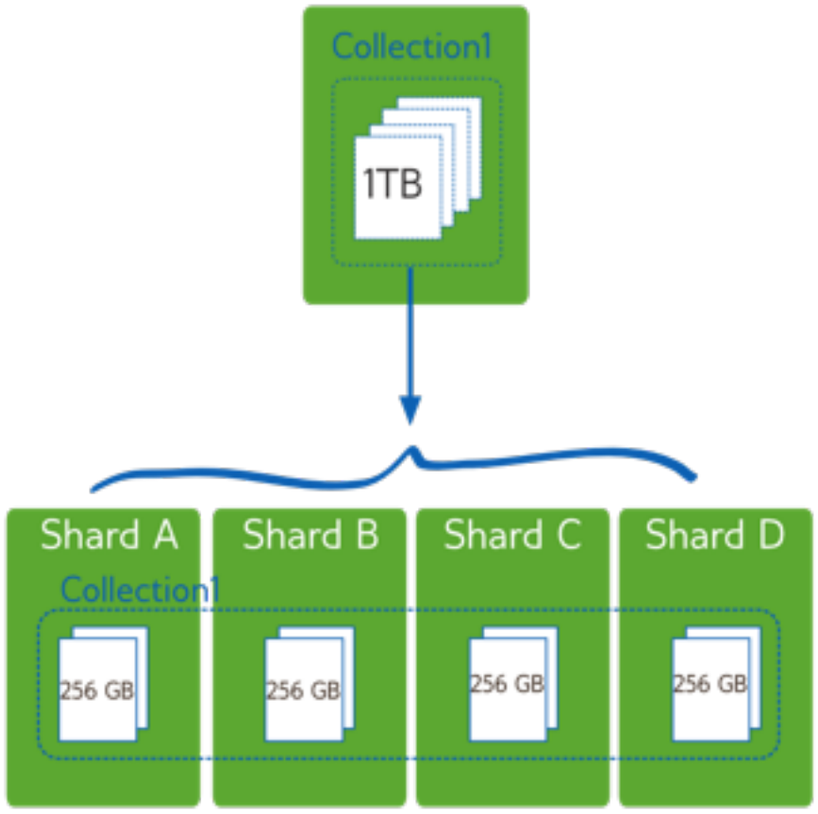
- To address these issues of scales, database systems have two basic approaches: vertical scaling and sharding.

  - **Vertical scaling** adds more CPU and storage resources to increase capacity. Scaling by adding capacity has limitations: high performance systems with large numbers of CPUs and large amount of RAM are disproportionately more expensive than smaller systems. Additionally, cloud-based providers may only allow users to provision smaller instances. As a result there is a practical maximum capability for vertical scaling.

  - **Sharding, or horizontal scaling**, by contrast, divides the data set and distributes the data over multiple servers, or shards. Each shard is an independent database, and collectively, the shards make up a single logical database.

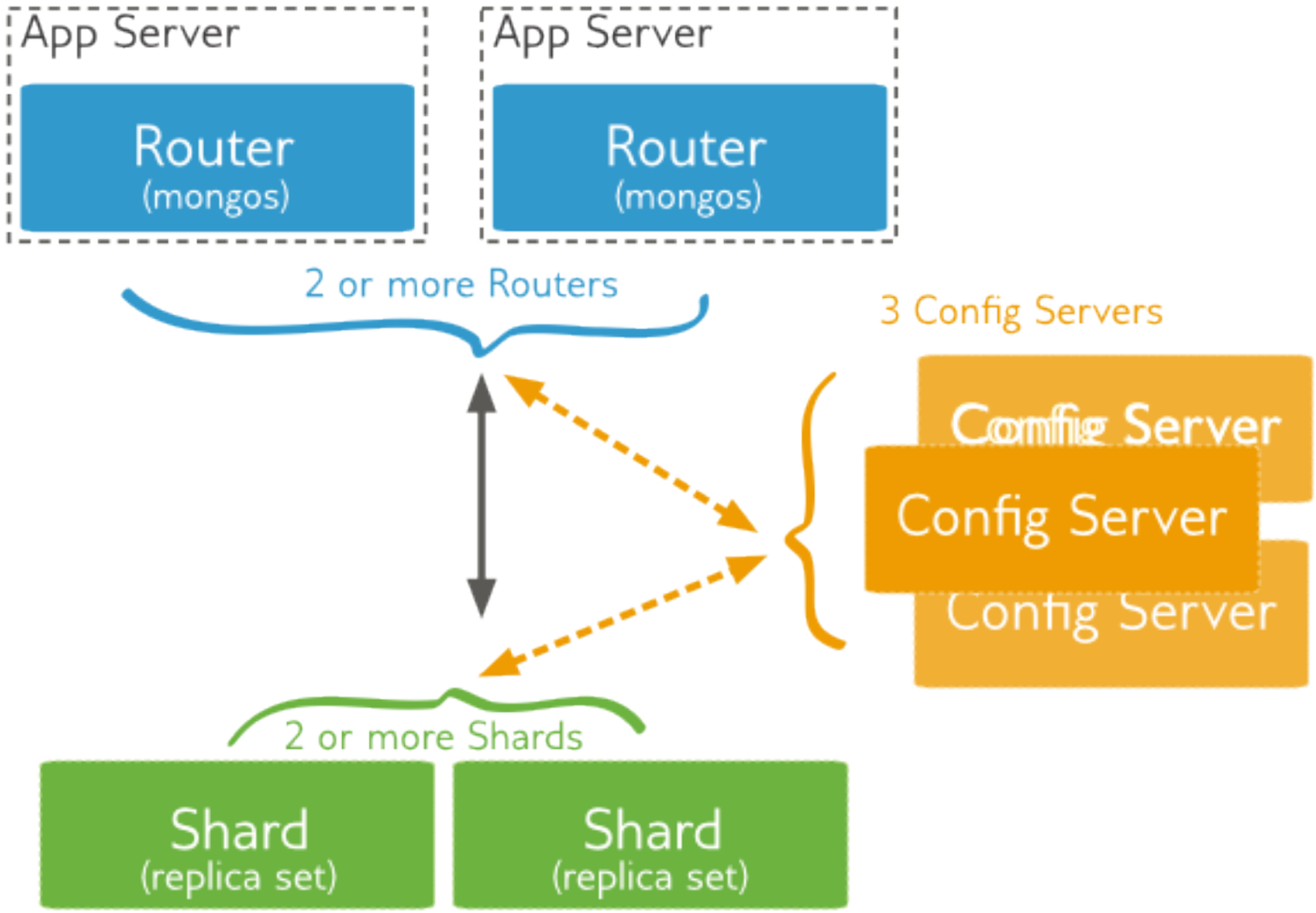
Sharding Architecture

- Sharding addresses the challenge of scaling to support high throughput and large data sets:
    - Sharding reduces the number of operations each shard handles. Each shard processes fewer operations as the cluster grows. As a result, a cluster can increase capacity and throughput horizontally.
      For example, to insert data, the application only needs to access the shard responsible for that record.
    - Sharding reduces the amount of data that each server needs to store. Each shard stores less data as the cluster grows.
      For example, if a database has a 1 terabyte data set, and there are 4 shards, then each shard might hold only 256GB of data. If there are 40 shards, then each shard might hold only 25GB of data.



Sharding in MongoDB

- MongoDB supports sharding through the configuration of a sharded clusters.



MongoDB Shard Components

- Sharded cluster has the following components: **shards**, **query routers** and **config servers**.

  - **Shards** store the data. To provide high availability and data consistency, in a production sharded cluster, each shard is a replica set.

  - **Query Routers**, or mongos instances, interface with client applications and direct operations to the appropriate shard or shards. The query router processes and targets operations to shards and then returns results to the clients. A sharded cluster can contain more than one query router to divide the client request load. A client sends requests to one query router. Most sharded clusters have many query routers.

  - **Config servers** store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards. Production sharded clusters have exactly 3 config servers.

Data Partitioning

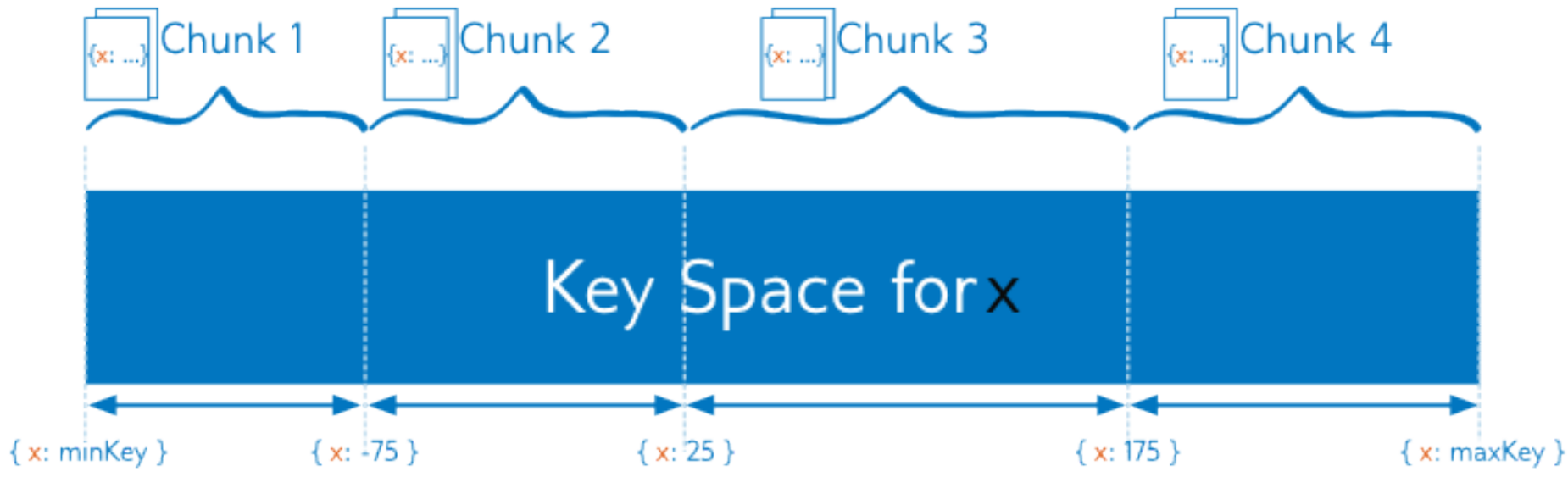
- MongoDB distributes data, or shards, at the collection level. Sharding partitions a collection's data by the shard key.

- To shard a collection, you need to select a **shard key**. A shard key is either an indexed field or an indexed compound field that exists in every document in the collection. MongoDB divides the shard key values into chunks and distributes the chunks evenly across the shards.

- To divide the shard key values into chunks, MongoDB uses either range based partitioning or hash based partitioning.

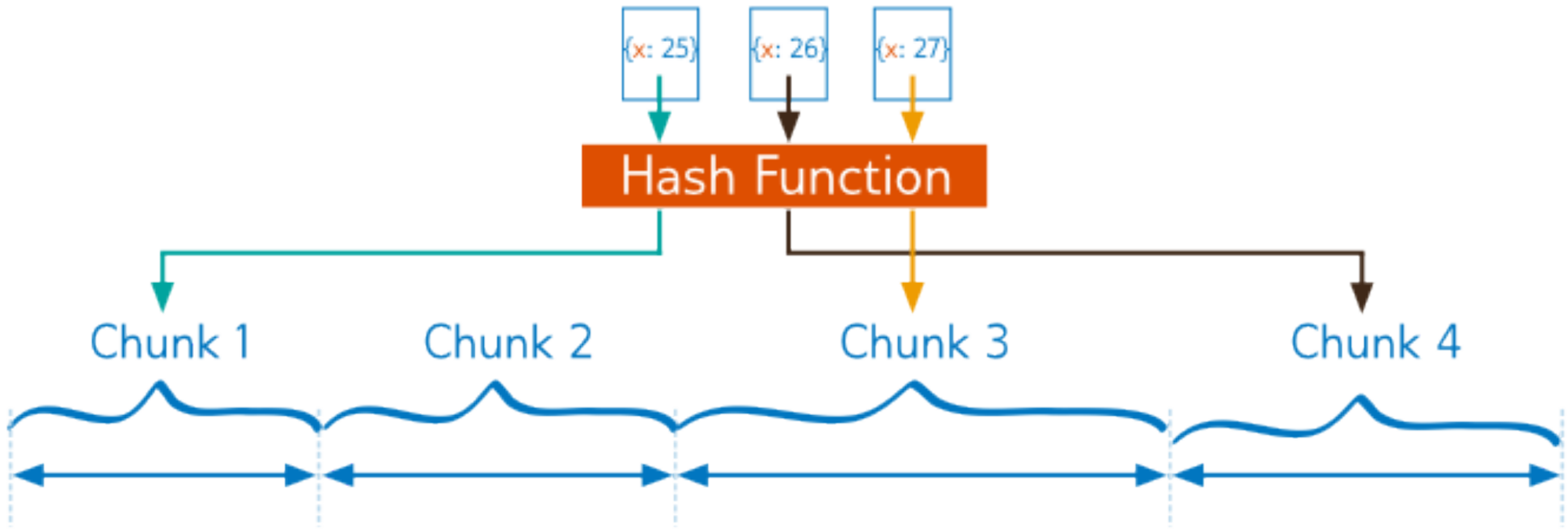
Range Based Sharding

- For range-based sharding, MongoDB divides the data set into ranges determined by the shard key values to provide range based partitioning.
- Consider a numeric shard key: If you visualize a number line that goes from negative infinity to positive infinity, each value of the shard key falls at some point on that line. MongoDB partitions this line into smaller, non-overlapping ranges called chunks where a chunk is range of values from some minimum value to some maximum value.
- Given a range based partitioning system, documents with “close” shard key values are likely to be in the same chunk, and therefore on the same shard.

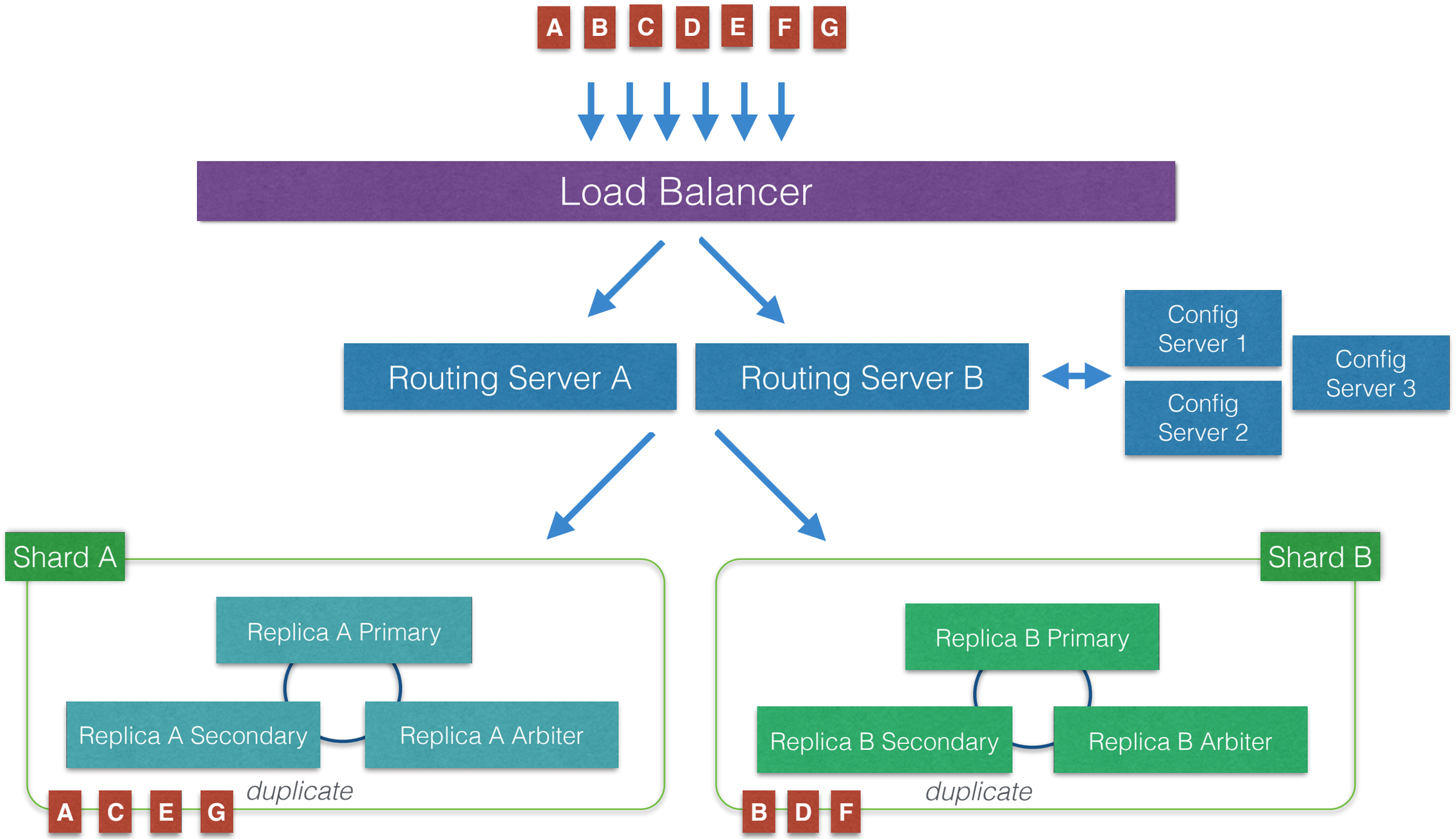


Hash Based Sharding

- For hash based partitioning, MongoDB computes a hash of a field's value, and then uses these hashes to create chunks.
- With hash based partitioning, two documents with "close" shard key values are unlikely to be part of the same chunk. This ensures a more random distribution of a collection in the cluster.



MongoDB Replica Set & Shard



g able

Performance Distinctions between Range and Hash Based Partitioning

- Range based partitioning supports more efficient range queries. Given a range query on the shard key, the query router can easily determine which chunks overlap that range and route the query to only those shards that contain these chunks.

- However, range based partitioning can result in an uneven distribution of data, which may negate some of the benefits of sharding.
    - For example, if the shard key is a linearly increasing field, such as time, then all requests for a given time range will map to the same chunk, and thus the same shard. In this situation, a small set of shards may receive the majority of requests and the system would not scale very well.

- Hash based partitioning, by contrast, ensures an even distribution of data at the expense of efficient range queries. Hashed key values results in random distribution of data across chunks and therefore shards. But random distribution makes it more likely that a range query on the shard key will not be able to target a few shards but would more likely query every shard in order to return a result.

Lab1: Prepare Shard Serves

1. Create 3 folders on the desktop, named “Shard1”, “Shard2”, and “Config”.

   Note: if you have and problems between walk-through this lab, and you want to start it over again. Just delete all files and folders under “Shard1”, “Shard2”, and “Config” folder.

2. Open Terminal. Issue the following command to stop default MongoDB instance.

   ```
   sudo service mongod stop
   ```

3. Issue the following command to start MongoDB instance for the first shard server.

   ```
   mongod --shardsvr --port 27027 --dbpath /home/mongo-root/Desktop/
   Shard1 --smallfiles --oplogSize 128
   ```

   Note: in this lab, we start shard server with port 21027, not default port, because mongos process (in step 6) will requires port 27017 to wait connection from clients.

Lab1: Prepare Config & Router

4. Open new Terminal. Repeat step 3 again with port 27018 to start the second shard server.

   ```
   mongod --shardsvr --port 27028 --dbpath /home/mongo-root/Desktop/
   Shard2 --smallfiles --oplogSize 128
   ```

5. Open the third Terminal. Issue the following command to start MongoDB instance for ConfigServer.

   ```
   mongod --configsvr --dbpath /home/mongo-root/Desktop/Config --
   port 27029 --smallfiles --oplogSize 128
   ```

   Note: The default port for config servers is 27019. You also can specify a different port.

6. Open the forth Terminal. Issue the following command to start MongoDB instance for Router Server.

   ```
   mongos --configdb 127.0.0.1:27029
   ```

Lab2: Enable Shard in Collection

1. Open the fifth Terminal. Issue the following command to connect to local mongos instance

   ```
   mongo
   ```

2. Add list of Shard Server to MongoDB Shard Configuration.

   ```
   sh.addShard("localhost:27027")
   sh.addShard("localhost:27028")
   ```

3. Configure database and collection to enable shard.

   ```
   use twitter_data
   sh.enableSharding("twitter_data")
   db.tweets.ensureIndex( {id : "hashed" })
   sh.shardCollection("twitter_data.tweets", {"id":"hashed"})
   ```

Lab3: Import data into Shard Cluster

1. Open Terminal. Goto desktop folder.

   cd desktop

2. Import twitter data into mongo cluster

   ```
   mongoimport --db twitter_data --collection tweets --file
   tweets.json
   ```

3. Login to mongos process to validate imported data.

   ```
   mongo

   show dbs              // you should see twitter_data database
   use twitter_data
   show collections      // you should see tweets collection
   db.tweets.count()     // write down the number of documents.
   ```

Lab2: Import data into Shard Cluster

4. Exit from mongos

   ```
   exit
   ```

5. Login to the first shard server to see the data. Write down how many documents in the tweets collection.

   ```
   mongo --port 27027
   show dbs
   use twitter_data
   db.tweets.count()
   ```

6. Repeat step 5 to login the second shard server. Write down how many documents in the tweets collection and check that documents is correct.

Summary & Tips

- As you can see in this lab. MongoDB will shard the data over shard server members to balance the documents load.

- You can start using Mongo as a single instance in the beginning then enable shard later when needs.

- Shard cluster also works with Replica sets, and its recommended to always enable replication.

- It’s not recommend to enable Shard cluster in the first place. Shard may not help if administrator defines keys that not match with document characteristics. Maximize the server’s configuration before sharding is recommended.

References

- Sharding Introduction
  http://docs.mongodb.org/manual/core/sharding-introduction/

- Tutorial & Deploy
  http://docs.mongodb.org/manual/tutorial/deploy-shard-cluster/