MongoDB Sharding

Kayartaya Vinod

Sharding

- Sharding is a method for distributing data across multiple machines.

- MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

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

    - For example, high query rates can exhaust the CPU capacity of the server.

    - Working set sizes larger than the system’s RAM stress the I/O capacity of disk drives.

Vertical scaling Vs Horizontal scaling

- Vertical Scaling involves increasing the capacity of a single server, such as using a more powerful CPU, adding more RAM, or increasing the amount of storage space.

- Limitations in available technology may restrict a single machine from being sufficiently powerful for a given workload.

- Additionally, Cloud-based providers have hard ceilings based on available hardware configurations.

- As a result, there is a practical maximum for vertical scaling.

Vertical scaling Vs Horizontal scaling

- Horizontal Scaling involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required.

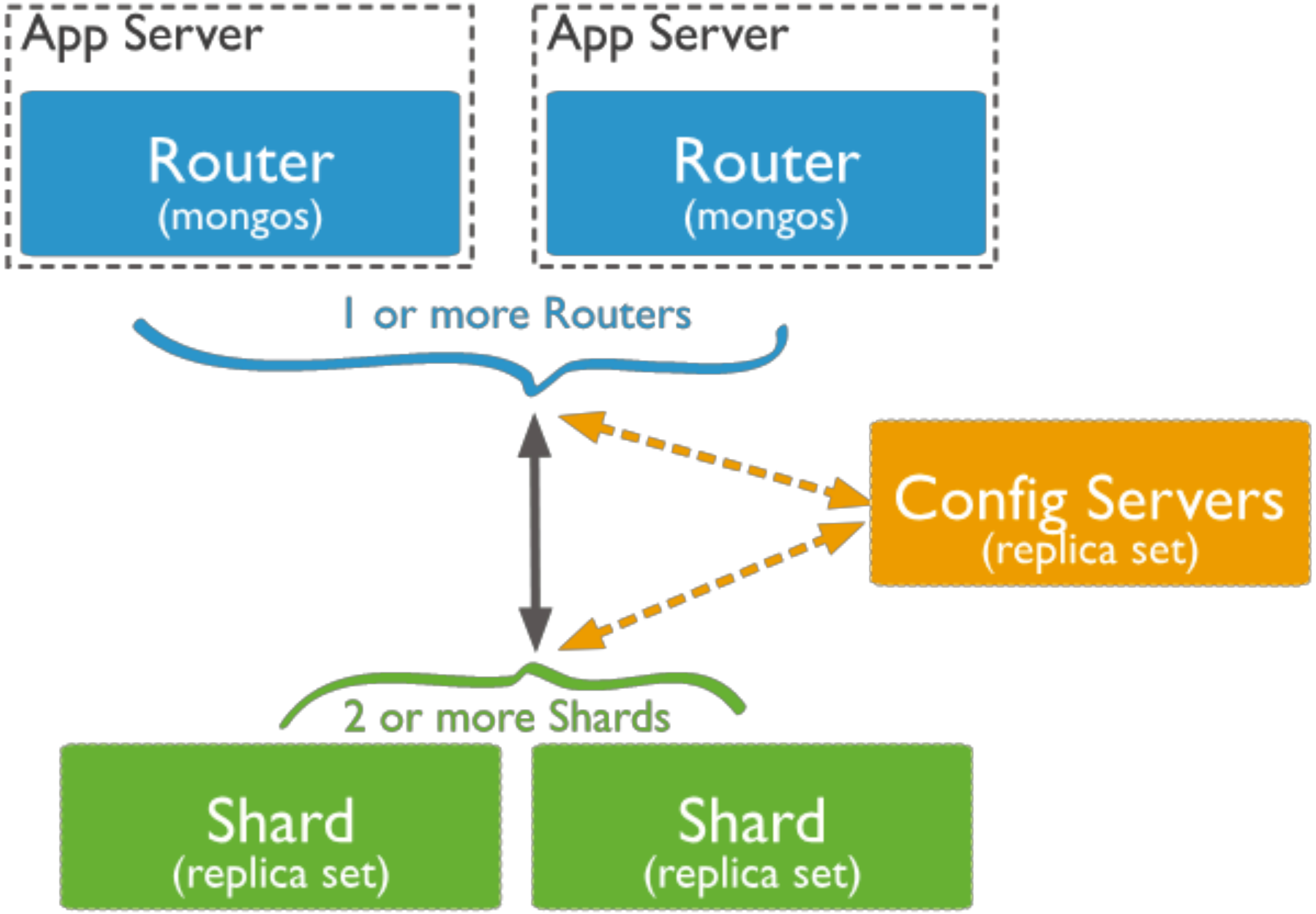
- While the overall speed or capacity of a single machine may not be high, each machine handles a subset of the overall workload, potentially providing better efficiency than a single high-speed high-capacity server.

- Expanding the capacity of the deployment only requires adding additional servers as needed, which can be a lower overall cost than high-end hardware for a single machine.

- The trade off is increased complexity in infrastructure and maintenance for the deployment.

Horizontal scaling in MongoDB

- MongoDB supports horizontal scaling through sharding.



Advantages of Sharding

- Reads / Writes
    - MongoDB distributes the read and write workload across the shards in the sharded cluster, allowing each shard to process a subset of cluster operations.
    - Both read and write workloads can be scaled horizontally across the cluster by adding more shards.
    - For queries that include the shard key or the prefix of a compound shard key, mongos can target the query at a specific shard or set of shards.
    - These targeted operations are generally more efficient than broadcasting to every shard in the cluster.

Advantages of Sharding

- Storage Capacity
  - Sharding distributes data across the shards in the cluster, allowing each shard to contain a subset of the total cluster data.
  - As the data set grows, additional shards increase the storage capacity of the cluster.

Advantages of Sharding

- High Availability
  - A sharded cluster can continue to perform partial read / write operations even if one or more shards are unavailable.
  - While the subset of data on the unavailable shards cannot be accessed during the downtime, reads or writes directed at the available shards can still succeed.

Sharded Cluster Components

- shard:
  - Each shard contains a subset of the sharded data.
  - As of MongoDB 3.6, shards must be deployed as a replica set.
- mongos:
  - The mongos acts as a query router, providing an interface between client applications and the sharded cluster.
- config servers:
  - Config servers store metadata and configuration settings for the cluster.
  - As of MongoDB 3.4, config servers must be deployed as a replica set (CSRS).

Production Configuration

- In a production cluster, ensure that data is redundant and that your systems are highly available.

- Consider the following for a production sharded cluster deployment:

  - Deploy Config Servers as a 3 member replica set

  - Deploy each Shard as a 3 member replica set

  - Deploy one or more mongos routers

Number of Shards

- Sharding requires at least two shards to distribute sharded data.

- Single shard sharded clusters may be useful if you plan on enabling sharding in the near future, but do not need to at the time of deployment.

Number of mongos and Distribution

- Deploying multiple mongos routers supports high availability and scalability.
  - A common pattern is to place a mongos on each application server.
  - Deploying one mongos router on each application server reduces network latency between the application and the router.

Number of mongos and Distribution

- Alternatively, you can place a mongos router on dedicated hosts.

- Large deployments benefit from this approach because it decouples the number of client application servers from the number of mongos instances.

- This gives greater control over the number of connections the mongod instances serve.

Number of mongos and Distribution

- Installing mongos instances on their own hosts allows these instances to use greater amounts of memory.

- Memory would not be shared with a mongod instance.

- It is possible to use primary shards to host mongos routers but be aware that memory contention may become an issue on large deployments.

Number of mongos and Distribution

- There is no limit to the number of mongos routers you can have in a deployment.

- However, as mongos routers communicate frequently with your config servers, monitor config server performance closely as you increase the number of routers.

- If you see performance degradation, it may be beneficial to cap the number of mongos routers in your deployment.

Shard Keys

- To distribute the documents in a collection, MongoDB partitions the collection using the shard key.

- The shard key consists of an immutable field or fields that exist in every document in the target collection.

Shard Keys

- You choose the shard key when sharding a collection.
- The choice of shard key cannot be changed after sharding.
- A sharded collection can have only one shard key.

Shard Keys

- To shard a non-empty collection, the collection must have an index that starts with the shard key.

- For empty collections, MongoDB creates the index if the collection does not already have an appropriate index for the specified shard key.

Shard Keys

- The choice of shard key affects the performance, efficiency, and scalability of a sharded cluster.

- A cluster with the best possible hardware and infrastructure can be bottlenecked by the choice of shard key.

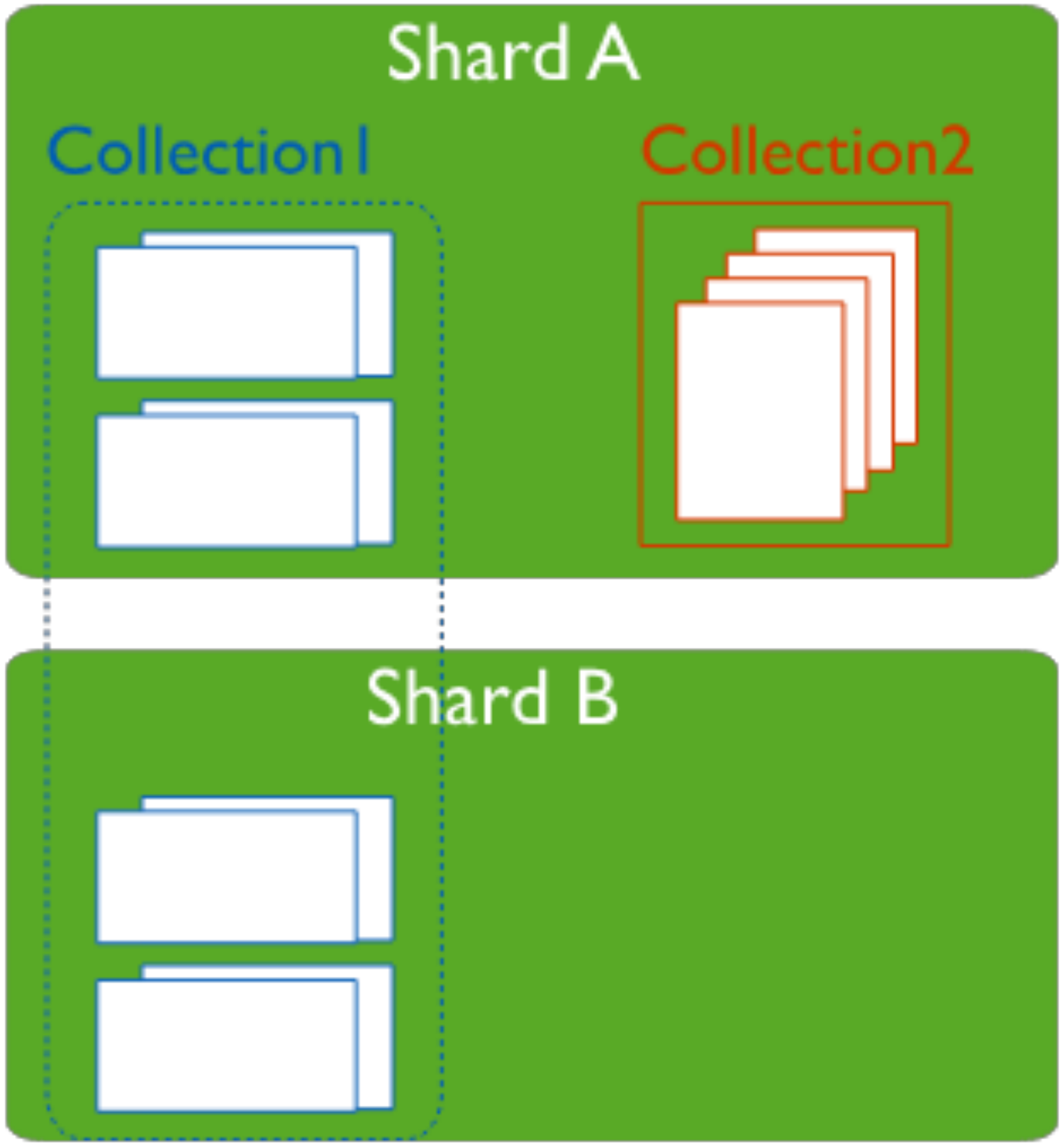
- The choice of shard key and its backing index can also affect the sharding strategy that your cluster can use.

Sharded and Non-Sharded Collections

- A database can have a mixture of sharded and unsharded collections.

- Sharded collections are partitioned and distributed across the shards in the cluster. Unsharded collections are stored on a primary shard.

- Each database has its own primary shard.

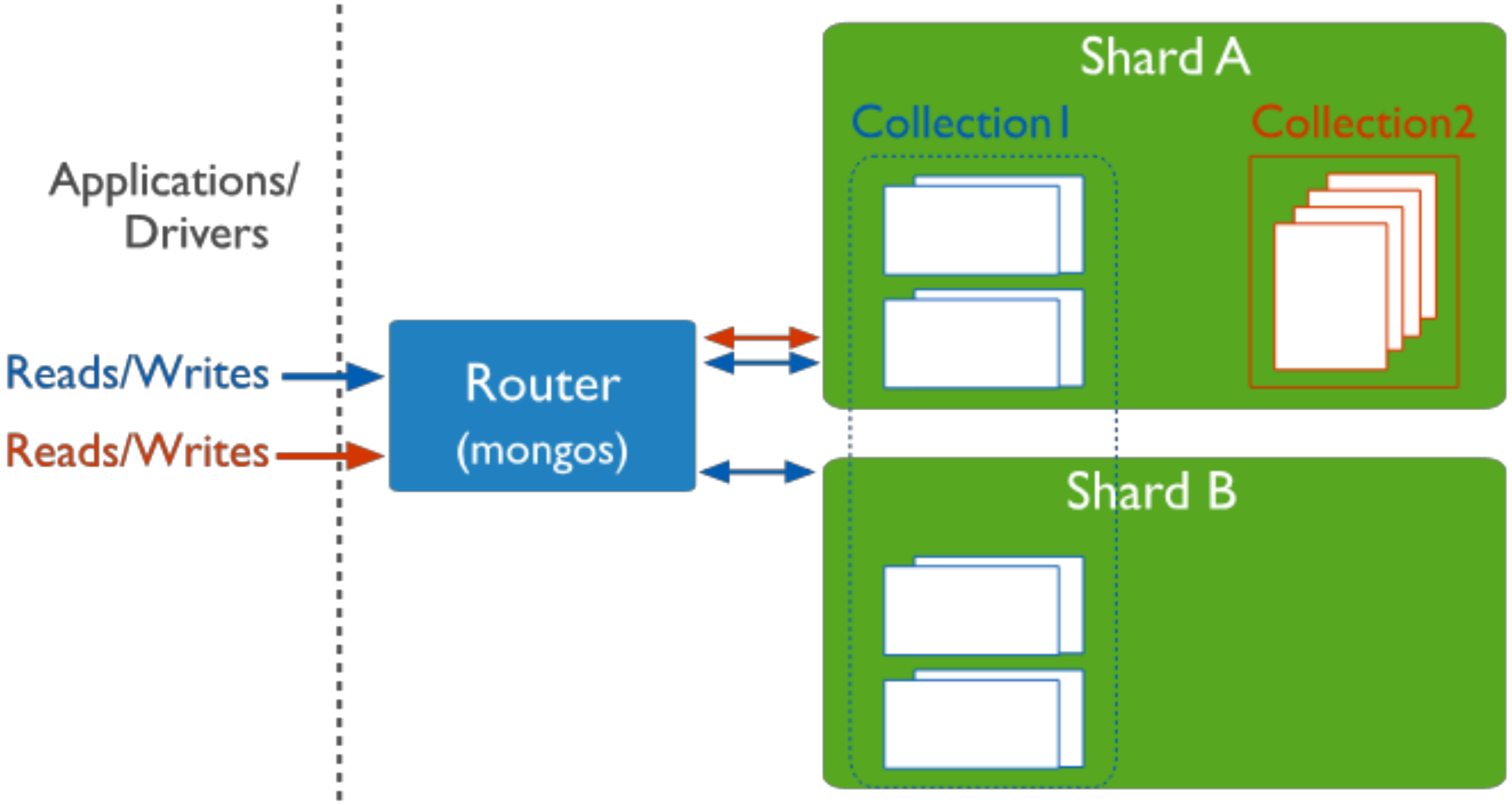


Connecting to a Sharded Cluster

- You must connect to a mongos router to interact with any collection in the sharded cluster.

- This includes sharded and unsharded collections.

- Clients should never connect to a single shard in order to perform read or write operations.



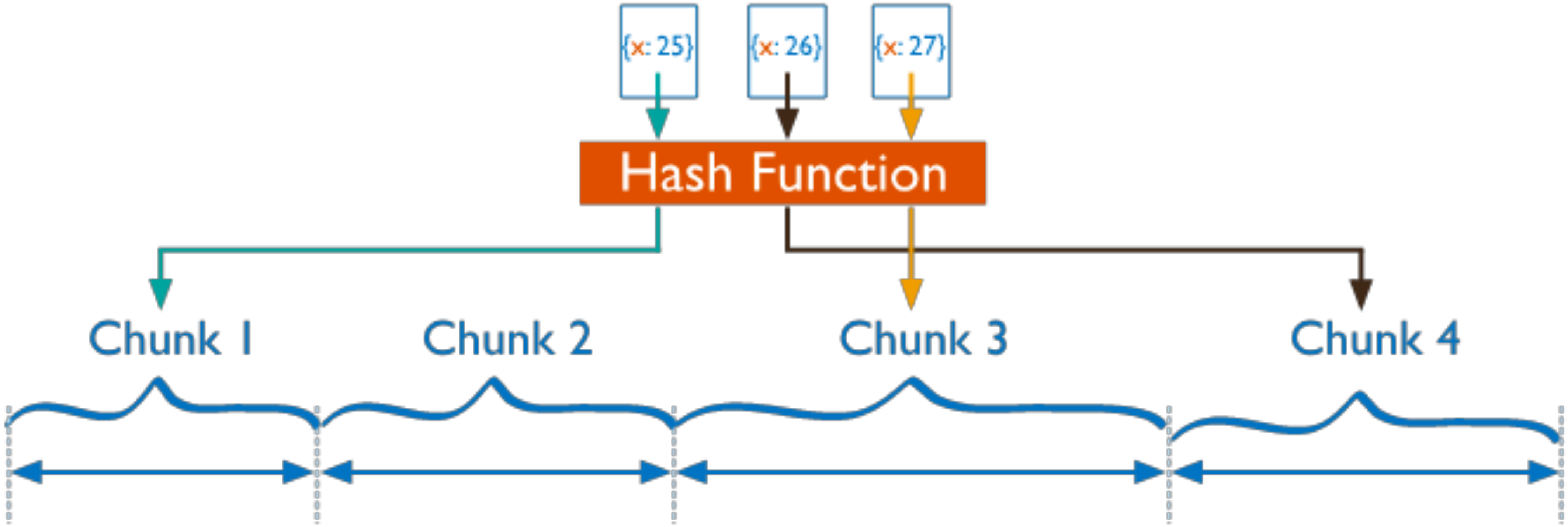
Sharding Strategy

- MongoDB supports two sharding strategies for distributing data across sharded clusters.
    - Hashed Sharding
    - Ranged Sharding

Hashed Sharding

- Hashed Sharding involves computing a hash of the shard key field’s value.

- Each chunk is then assigned a range based on the hashed shard key values.



Hashed Sharding

- While a range of shard keys may be “close”, their hashed values are unlikely to be on the same chunk.

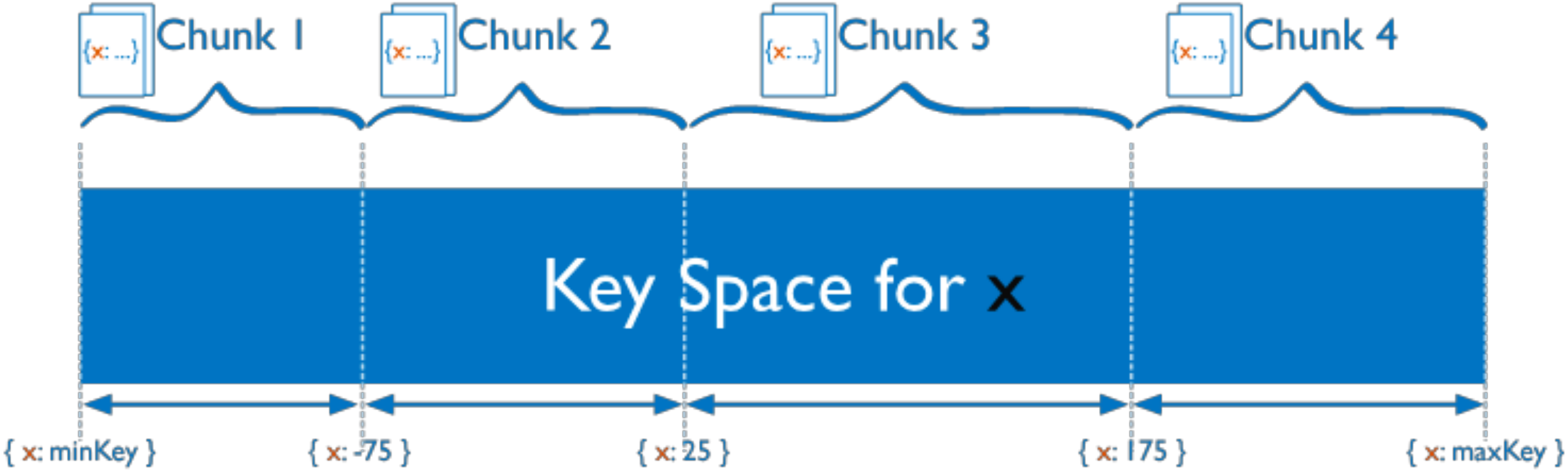
- Data distribution based on hashed values facilitates more even data distribution, especially in data sets where the shard key changes monotonically.

- However, hashed distribution means that ranged-based queries on the shard key are less likely to target a single shard, resulting in more cluster wide broadcast operations

Ranged Sharding

- Ranged sharding involves dividing data into ranges based on the shard key values.

- Each chunk is then assigned a range based on the shard key values.



Ranged Sharding

- A range of shard keys whose values are “close” are more likely to reside on the same chunk.

- This allows for targeted operations as a mongos can route the operations to only the shards that contain the required data.

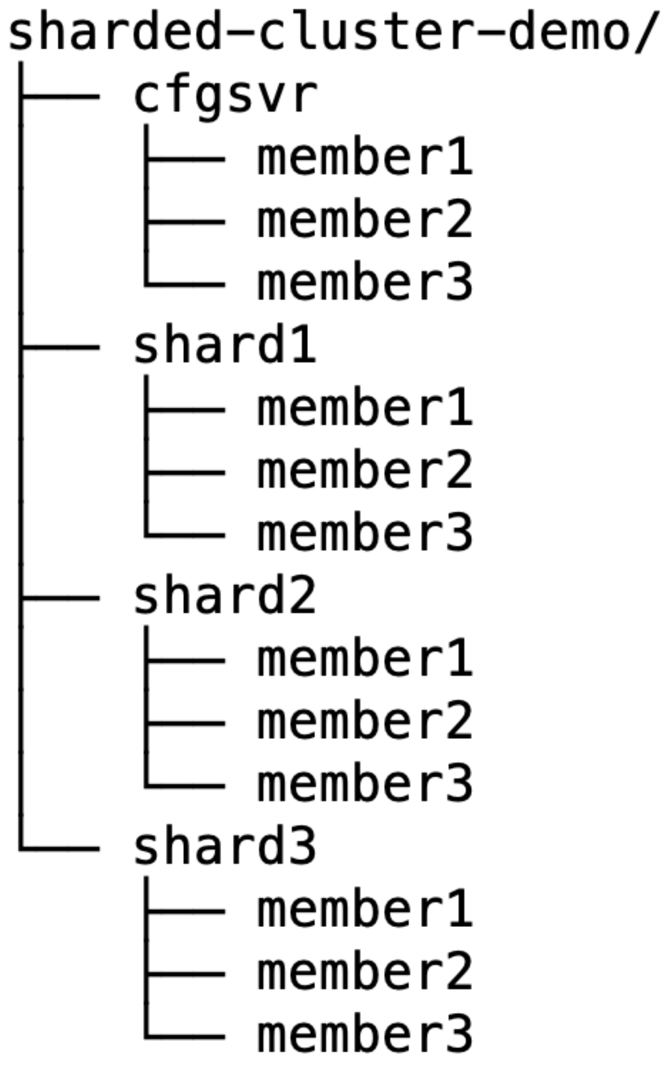
- The efficiency of ranged sharding depends on the shard key chosen.

- Poorly considered shard keys can result in uneven distribution of data, which can negate some benefits of sharding or can cause performance bottlenecks.

Sharded Cluster - How to?

- Create the folders for config servers and replica set members of all the shards

```
mkdir cfgsvr
mkdir shard1
mkdir shard2
mkdir shard3
mkdir cfgsvr/member1
mkdir cfgsvr/member2
mkdir cfgsvr/member3
mkdir shard1/member1
mkdir shard1/member2
mkdir shard1/member3
mkdir shard2/member1
mkdir shard2/member2
mkdir shard2/member3
mkdir shard3/member1
mkdir shard3/member2
mkdir shard3/member3
```



```
sharded-cluster-demo/
├── cfgsvr
│   ├── member1
│   ├── member2
│   └── member3
├── shard1
│   ├── member1
│   ├── member2
│   └── member3
├── shard2
│   ├── member1
│   ├── member2
│   └── member3
└── shard3
    ├── member1
    ├── member2
    └── member3
```

Sharded Cluster - How to?

- Start the MongoDB servers for config servers

```
mongod --configsvr --replSet cfg --port 26000 \
    --dbpath cfgsvr/member1 \
    --bind_ip localhost,192.168.31.78


mongod --configsvr --replSet cfg --port 26001 \
    --dbpath cfgsvr/member2 \
    --bind_ip localhost,192.168.31.78


mongod --configsvr --replSet cfg --port 26002 \
    --dbpath cfgsvr/member3 \
    --bind_ip localhost,192.168.31.78
```

Sharded Cluster - How to?

- Using a mongo shell, connect to the intended primary of the config server’s admin database and initiate the replica set

```
mongo --host 192.168.31.78 --port 26000

rs.initiate(
  {
    _id: "cfg",
    configsvr: true,
    members: [
      { _id : 0, host : "192.168.31.78:26000" },
      { _id : 1, host : "192.168.31.78:26001" },
      { _id : 2, host : "192.168.31.78:26002" }
    ]
  }
)
```

Sharded Cluster - How to?

- Start the MongoDB servers for config servers

```
mongod  --shardsvr --replSet shard1rs --port 27000 \
   --dbpath shard1/member1 \
   --bind_ip localhost,192.168.31.78

mongod  --shardsvr --replSet shard1rs --port 27001 \
   --dbpath shard1/member2 \
   --bind_ip localhost,192.168.31.78

mongod  --shardsvr --replSet shard1rs --port 27002 \
   --dbpath shard1/member3 \
   --bind_ip localhost,192.168.31.78
```

Sharded Cluster - How to?

- Using a mongo shell, connect to the intended primary of this shard’s admin database and initiate the replica set

```
mongo --host 192.168.31.78 --port 27000

rs.initiate(
  {
    _id: “shard1rs”,
    members: [
      { _id : 0, host : "192.168.31.78:26000" },
      { _id : 1, host : "192.168.31.78:26001" },
      { _id : 2, host : "192.168.31.78:26002" }
    ]
  }
)
```

Sharded Cluster - How to?

- Repeat the same for all the other shards (shard2rs, shard3rs)

Sharded Cluster - How to?

- Start the router (mongos) associating the config servers
- This one we are going to run on the default port

```
mongos \
--configdb cfg/192.168.31.78:26000,192.168.31.78:26001,192.168.31.78:26002 \
--bind_ip localhost,192.168.31.78
```

- cfg/<shard1primary>,<shard2primary>,<shard3primary>

Sharded Cluster - How to?

- Connect to the mongos to add shards, enable shards and shard collections

```
mongo --host 192.168.31.78 admin


sh.addShard( "a/192.168.31.78:27000")
sh.addShard( "b/192.168.31.78:27100")
sh.addShard( "c/192.168.31.78:27200")


sh.enableSharding("mydb")


sh.shardCollection(“mydb.customers", { _id : "hashed" } )
```