Cloud Computing and Big Data

LIS 4102 SECTION 1

Dr. Genoveva Vargas-Solar

Assignment 2: Hands-On MongoDB Sharding

Melina Escobedo Zárate

ID 164094

February 20th ‘22

Which is the important information reported by sh.status()?

The Sharding Version section displays all the information on the config database, then we have the Shards section, in which displays a list of all shard(s). For each shard we're able to know the name, host, if if has any tag, after that we have the Balancer section, here is a list of all the information about the state of the balancer. This provide us insight into current balancer operation and is useful when troubleshooting an unbalanced shared cluster, the last section is Database, in here all the information on the database(s) is listed. For each database is displays the name, whether the database has Sharding enable, and the primary shard for the database.

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("621146130fed99a610901c6e")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1.docker:27017" }
  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
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
```

Describe in natural language the content of the collection

Inside the collection we could find many brackets in which the first data is a unique id per each block of information that is used to identify, then we have the name of the city, and then latitude or longitude that are used for the location, from this we could find the population and the last line is the nomenclature of the state.

```
{
        "_id" : "01001",
        "city" : "AGAWAM",
        "loc" : [
                -72.622739,
                42.070206
        ],
        "pop" : 15338,
        "state" : "MA"
}
```

How many chunks did you create? Which are their associated ranges?

We created just 1 chunk with a range from 1 to 1

```
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("621146130fed99a610901c6e")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1.docker:27017" }
  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
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard1" }
                mydb.cities1
                        shard key: { "state" : 1 }
                        chunks:
                                shard1  1
                        { "state" : { "$minKey" : 1 } } -->> { "state" : { "$maxKey" : 1
} } on : shard1 Timestamp(1, 0)
```

How many chunks are there now? Which are their associated ranges? Which changes can you identify in particular?

Now we have 3 which a range 1 to 1. The Changes are just this new chunks with their associated information, as well as a change with the Timestamp

```
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("621146130fed99a610901c6e")
}
  shards:
        {  "_id" : "shard1",  "host" : "shard1.docker:27017" }
  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
  databases:
        {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
        {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard1" }
                mydb.cities1
                        shard key: { "state" : 1 }
                        chunks:
                                shard1  3
                        { "state" : { "$minKey" : 1 } } -->> { "state" : "MA" } on : shar
d1 Timestamp(1, 1)
                        { "state" : "MA" } -->> { "state" : "RI" } on : shard1 Timestamp(
1, 2)
                        { "state" : "RI" } -->> { "state" : { "$maxKey" : 1 } } on : shar
d1 Timestamp(1, 3)
```

How many chunks did you create? What differences do you see with respect to the same task in the range sharding strategy?

We have 3 chunks in the first collection and 2 in the second collection. It also have some changes in the information of the second collection, because it displays an NumberLong attribute.

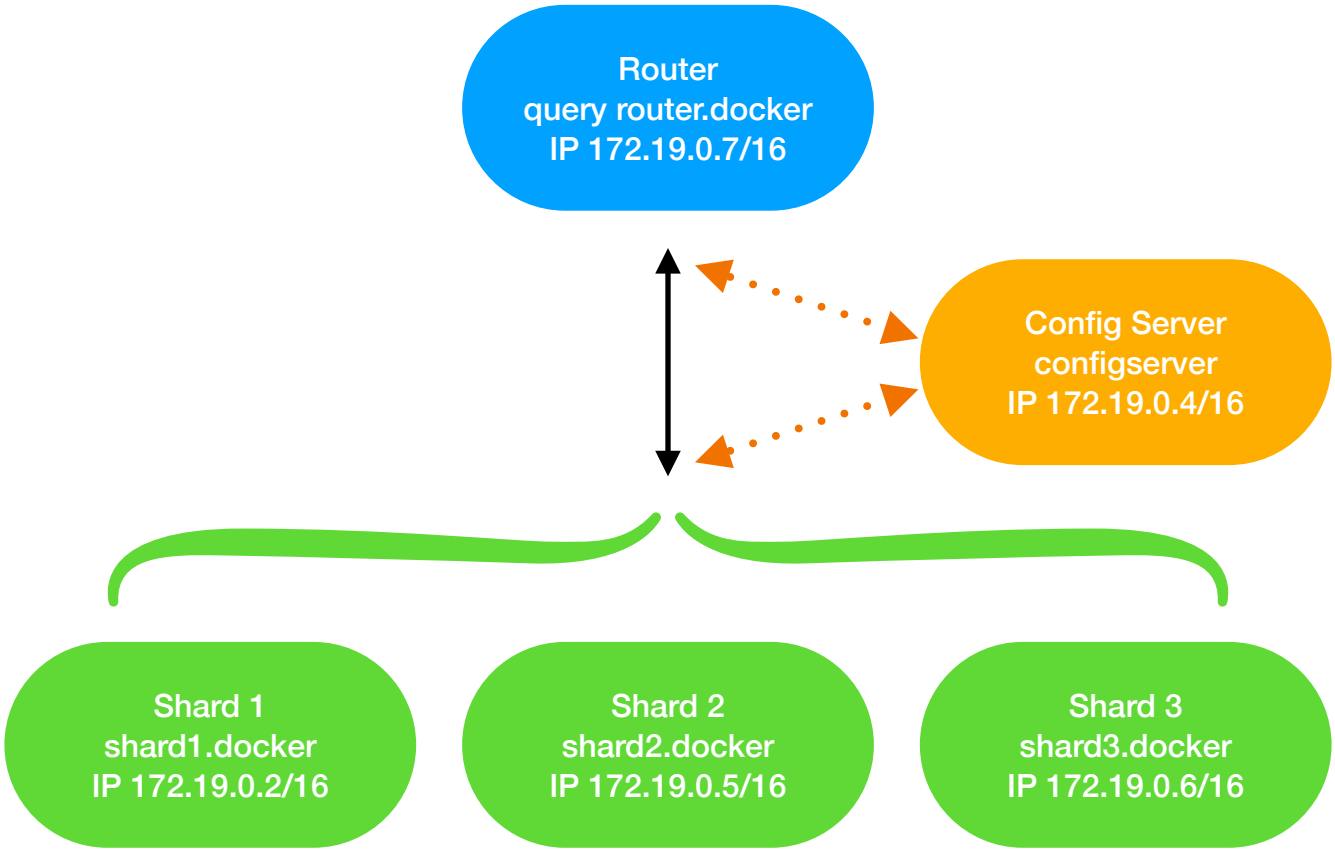
```
                mydb.cities1
                        shard key: { "state" : 1 }
                        chunks:
                                shard1  3
                        { "state" : { "$minKey" : 1 } } -->> { "state" : "MA" } on : shar
d1 Timestamp(1, 1)
                        { "state" : "MA" } -->> { "state" : "RI" } on : shard1 Timestamp(
1, 2)
                        { "state" : "RI" } -->> { "state" : { "$maxKey" : 1 } } on : shar
d1 Timestamp(1, 3)
                mydb.cities2
                        shard key: { "state" : "hashed" }
                        chunks:
                                shard1  2
                        { "state" : { "$minKey" : 1 } } -->> { "state" : NumberLong(0) }
on : shard1 Timestamp(1, 1)
                        { "state" : NumberLong(0) } -->> { "state" : { "$maxKey" : 1 } }
on : shard1 Timestamp(1, 2)
```

How many chunks are there now? Compare the result with respect to the range sharding.

We have 3 chunks in the first collection and 4 in the second collection. For the second collection its third and fourth chunk doesn't have a null value on their NumberLong attribute.

```
                mydb.cities1
                        shard key: { "state" : 1 }
                        chunks:
                                shard1  3
                        { "state" : { "$minKey" : 1 } } -->> { "state" : "MA" } on : shar
d1 Timestamp(1, 1)
                        { "state" : "MA" } -->> { "state" : "RI" } on : shard1 Timestamp(
1, 2)
                        { "state" : "RI" } -->> { "state" : { "$maxKey" : 1 } } on : shar
d1 Timestamp(1, 3)
                mydb.cities2
                        shard key: { "state" : "hashed" }
                        chunks:
                                shard1  4
                        { "state" : { "$minKey" : 1 } } -->> { "state" : NumberLong(0) }
on : shard1 Timestamp(1, 1)
                        { "state" : NumberLong(0) } -->> { "state" : NumberLong("36301929
31154748514") } on : shard1 Timestamp(1, 3)
                        { "state" : NumberLong("3630192931154748514") } -->> { "state" :
NumberLong("8134625179139298768") } on : shard1 Timestamp(1, 4)
                        { "state" : NumberLong("8134625179139298768") } -->> { "state" :
{ "$maxKey" : 1 } } on : shard1 Timestamp(1, 5)
```

Draw the new configuration of the cluster and label each element (router, config server and shards) with its corresponding IP



Analyze the results and explain the logic behind this tagging strategy. Connect to the shard that contains the data about California, and count the documents. Do the same operation with the other shards. Is the sharded data collection complete with respect to initial one? Are shards orthogonal?

The tag **sh.addTagRange** is to ensure that the balancer migrates documents that exist within the specified range to a specific shard or set of shards. This let us attached a range of shard key values to a shard tag. For each shard we have different number of documents, ion shard1 we have 1516, for shard2 we have1595 and for shard3 they are 26242. Well, at the very beginning we just had one collection, when we start making the activity we ended whit three more (cities1, cities2, cities3). I think they are orthogonal, because the documents doesn't repeat between shards.

Vademecum

What is a sharding key? Is the choice of a sharding key directly dependent of the sharding strategy? Explain and give examples.

The shard key is either a single indexed field or multiple fields covered by a compound index that determines the distribution of the collection's documents among the cluster's shards (A single MongoDB instance or replica set that stores some portion of a shared cluster's total data set. In production, all shard should be replica sets). In more simple words a shad key is the field MongoDB uses to distribute documents among members of a shared cluster. What MongoDB does, is divide the span of shard key values into non-overlapping ranges of shard key values, therefore each range should be associated with a chunk (A contiguous range of shard key values within a particular shard), this is, because MongoDB attempts to distribute chunks evenly among the shard in the cluster.
MongoDB just support two major sharding strategies for distributing data across shared clusters, but we have three:

- The Hashed Sharding involves computing a hash of the key field's value. It means that each chunk is then assigned a range based on the hashed shared key values. The thing is that while a range of shard keys may be "close", their hashed values are unlikely to be on the same chunk. However, hashed distribution means that range-bases queries on the shard key are less likely to target a single shard, resulting in more cluster wide broadcast operations.

- Know we have the Ranged Sharding that involves dividing data into ranges based on the shard values. Each chunk is then assigned a range based on the start key values. In this case a range of shard keys whose values are "close" are more likely to reside on the same chunk. It means, this allows for targeted operations as a mongos can route the operations to only the shard that contains the required data, and depend on the shard key is chosen.

- Zoned Sharding is the one that provides the ability for developers to define specific rules governing data placement in a sharded cluster.

The shard key has a direct relationship to the effectiveness of chunk distribution.

Explain how does apparently MongoDB chooses the number of intervals used to shard a collection in the Interval oriented strategy? In which situations would such a strategy be well adapted for sharding a collection?

First of all, the default number of chunks size for a shared cluster is 64 megabytes, this is the default size for splitting and migrating chunks. In MongoDB the allowed range of the chunk size is between 1 and 1024 megabytes.
To choose the number of intervals used to shard a collection, MongoDB evaluates if its a Ranged, Hashed or Zoned Sharding. Because if's a ranged, the documents are partitioned across shards according to the shard key value, this strategy is well suited for applications that could need to optimize range bases queries, like co-locating data for all customers in just an specific region or shard. Meanwhile in hashed the documents are distributed according to an MD5 has of the shad key value, the uniform distributions of writes across shards is guaranteed. And if it is a zoned, the developer is the one with the power to define rules.

To adapt a specific strategy, we need to take into account that data can be distributed according to query patterns or data placement requirements, that leads to a higher scalability.

In which situations would the hash-based strategy be interesting for a collection to be sharded?

For example, for applications that issue range-based queries, ranged Sharding is beneficial because operations can be routed to the fewest shards necessary, usually in a single shard. However, ranges Sharding required a good and clear understanding of data and query patterns, which in some cases may not be practical. But with hashed Sharding strategy we ensure a uniforms distribution of reads and writes, but it doesn't provide efficient range-base operations. It would be interesting to use Hash-Based because MongoDB will do pretty much all the hard work, this means that it would randomize data distribution based on whatever kind of document identifier we decide. So this means that as long as the identifier has a high cardinality, all of the documents in the collection shall be spread evenly across the shards of the cluster. If we think deeper, for a workload of individuals documents, this would be the best choice.

Which of the strategies interval or sharding would lead to a more balanced distribution of data across shards, interval, or hash?

Maybe we don't need to choose one, because MongoDB contains a balancer, that is a background process that is in charge of monitoring the number of chunks on each shard. This means that whenever the number of chunks on a given shard reaches a specific migration threshold, the balancer attempt to automatically migrate chunks between shards and each equal number of chunks per shard. In more simple words, this is an entirely transparent procedure that balance the data automatically. Btw hash could be the best option, because its values are better for a more efficient distribution.

What are the advantages and disadvantages of allowing access to shards directly through their server and not only through the query router?

Some disadvantages are that direct routing is the preferred way of accessing shards to achieve better performance, among other benefits. Another big advantage of query routing is that it offers better performance, accommodates geographic distribution of shards, eases load balancing and supports all type of queries. Adding to all this, this is just used to perform administrative or maintenance moves, so if our systems is not well designed, it could crash.

Give an example of situation where tag based sharding would be an interesting option?

An obvious example is when we need to create and adjust policies that affect where documents are stored based on user-defined tags and tag range. It's an interesting option when we need to maximize the performance on the cluster, or optimize storage, also to implement data center-awareness in the application. For example, a Bank use to needs to store data that is not frequently accessed (transactions details or claims for a number of years to stay in compliance with governments regulations) and this is very expensive to keep for several years, but if we use Tag Sharding we could assign tags to different storages tiers and associate a unique shard key range to a tag, and the documents shall move from one shard to the next during normal balancing operations. This is great because it still keep all the data in one database, but we don't make it expensive.

What happens when a new shard is added to a cluster containing already other shards with data?

Nothing bad, because when we add a new shard to an existing cluster, it automatically would become the shard with the lowest number of chunks for every shared collection. It means that it will be the default target for migration from the shard with the highest number of chunks until the balancer do its job. But in the other hand, if we added the traffic would increase. And migration would take a while, especially if things are under load.

How would you test whether a sharded collection was an interesting solution in comparison to a centralized one?

I think this would be more into experience, because if we have a favorable result it would be the best option for us, but if not, we know the answer. It is important to analyze in what situation and conditions does the shared collection worked, because if we want to do changes at least we could have an option to return to the begging knowing the “base” conditions for a better workspace. Once we have this, we could experiment with other options, and again analyze if they are optimal conditions to work in. It’s also important to know with which type of data we're managing, we can't use something we have used if the data needs something else to be manipulated.