CSCI235 Database Systems

MongoDB Sharding

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Sharding

Outline

- Basics
- Why sharding ?
- Architecture
- Distribution
- Experiment
- Querying and indexing
- Production

Basics

Sharding is the process of partitioning a large dataset into smaller and more manageable pieces

A shared nothing architecture is a distributed computing architecture in which each computing node does not share data with other the nodes

In database systems it is called as sharding (shared nothing)

What to do when:

- large amount of data and greater read/write throughput demands make commodity database servers not sufficient ?
- the database servers are not be able to address enough RAM, or they might not have enough CPU cores, to process the workload efficiently ?
- due to large amount of data it is not practical to store and to manage backups on one disk or RAID storage (Redundant Array of Inexpensive Disks) ?

Basics

A solution to these problems is to distribute a database and database processing across more than one server

The method for doing this in Mongo DB is called sharding.

Sharding makes a database system complex due to administrative and performance overhead

Sharding

Outline

- Basics
- Why sharding ?
- Architecture
- Distribution
- Experiment
- Querying and indexing
- Production

Why sharding ?

There are two main reasons to shard:

- storage distribution
- load distribution

If monitoring of storage capacity shows that at certain moment the database applications require more storage than it is available and adding more strage is impossible then sharding is the best option

Mongodb monitoring means running db.stats() and db.collection.stats() in the mongo shell to get the statistics about the storage usage of the current database and collection within it

Load means CPU and RAM utilization, I/O bandwidth, network transmission used by the requests from the clients => response time

If at certain moment response time does not match the client's expectations then it triggers a decision to shard

A decision to shard depends on network usage, disk usage, CPU usage, and RAM usage

Sharding

Outline

- Basics
- Why sharding ?
- Architecture
- Distribution
- Experiment
- Querying and indexing
- Production

Architecture

In MongoDB a sharded cluster consists of shards, mongos routers, and config servers

Shards store the application data

mongos routers or system administrators are connected directly to the shards

A shard is either a single mongod server or a replica set that stores a partition of the application data

Like an unsharded deployment, each shard can be a single node for development and testing stage

Each shard should be a replica set in production stage because it is able to provide automatic replication and failover mechanisms

Shards are the only places where the application data gets saved in a sharded cluster

Architecture

It is possible to connect to an individual shard as to a single computing node or a replica set, then it is possible to see only a portion of the total data stored in entire shard

Because each shard contains only part of entire data it is necessary to route the operations to the appropriate shards

mongos routers cache the cluster metadata and use it to route operations to the correct shard or shards

mongos routers provide clients with a single point of contact with the cluster

mongos provide a view of a sharded cluster the same as a view an unsharded one

Because mongos processes are lightweight and nonpersistent they can be deployed on the same machines as the application servers

Then only one network step is required for requests to any given shard

Architecture

As mongos processes are non-persistent then an additional process is needed to durably store the metadata needed to properly manage the cluster

Config servers persistently store metadata about the cluster, including which shard has what subset of the data

Metadata includes the global cluster configuration, the locations of each database, collection, and the particular ranges of data in a collection and a change log preserving a history of the migrations of data across shards

For example, every time a mongos process is started, it fetches a copy of the metadata from config servers to get a coherent view of the shard cluster

Shard cluster requires several config servers not deployed as as a replica set

Write operations on a config server use a two-phase commit protocol to ensure the data consistency across the config servers

Architecture

In production environment more than one config server must be used and all config servers must reside on the separate machines to provide redundancy

Sharding

Outline

- Basics
- Why sharding ?
- Architecture
- Distribution
- Experiment
- Querying and indexing
- Production

Distribution

There are four levels of data granularity in MongoDB:

- Document: the smallest unit of data in MongoDB; a document represents a single object in the system (like a row in a relational database)
- Chunk: a group of documents clustered by the values on a field; a chunk is a concept that exists only in sharded setups; a chunk is created by the logical grouping of documents based on their values for a key or set of keys, known as a shard key
- Collection: a named grouping of documents within a database; a collection allows users to separate a database into logical groupings that make sense for the application
- Database: a set of collections of documents; a combination of a database name and a collection name is unique throughout the system, and it is commonly referred to as the namespace

Distribution

Data can be distributed in a sharded cluster in the following ways:

- at a level of an entire database where each database along with all its collections is put on its own shard
- at a level of partitions or chunks of a collection, where the documents within a collection itself are spread out over multiple shards based on values of a key or set of keys (shard key) in the documents

Database Distribution

Each database in a sharded cluster is assigned to a different shard

A database itself is not sharded

It is some sort of manual sharding (partitioning)

MongoDB has nothing to do with how well data is partitioned and it is completely up to a user to decide which database is located into which shard

One example of a real application for database distribution is MongoDB as a service

In such implementation of sharding customers can pay for access to a single MongoDB database

Collection Distribution

The second method is sharding an individual collection

It is an automatic sharding in which MongoDB itself makes all the partitioning decisions, without any direct intervention from the applications

For example, consider the following document:

BSON
```
{
  "_id": ObjectId("4d6e9b89b600c2c196442c21")
  "filename": "spreadsheet-1",
  "updated_at": ISODate("2011-03-02T19:22:54.845Z"),
  "username": "banks",
  "data": "raw document data"
}
```

If all the documents in a collection have this format, and if we choose `"_id"` key and the `"username"` key as a shard key then a pair of values associated with `"_id"` and `"name"` in each document is used to determine what chunk the document belongs to

Collection Distribution

Sharding in MongoDB is range-based

It means that each chunk represents a range of shard keys

To determine what chunk a document belongs to, MongoDB extracts the values for a shard key and then finds a chunk whose shard key range contains the given shard key values

Sharding

Outline

- Basics
- Why sharding ?
- Architecture
- Distribution
- Experiment
- Querying and indexing
- Production

Experiment

A process of setting up a sharded cluster consists of three steps:

- starting the mongod and mongos servers through spawning all the individual mongod and mongos processes that make up the cluster
- configuring the cluster through updating the configuration such that the replica sets are initialized and the shards are added to the cluster and the nodes are be able to communicate with each other
- sharding collections such that it can be spread across multiple shards

Start Terminal and process the following shell commands to create config server (just one)

```
mkdir conf1
mkdir conf2
```

Command shell

Process the following command in a new Terminal window

```
mongod --configsvr --replSet conf --dbpath conf1 --port 4001
```

Command shell

Experiment

Process the following command in a new Terminal window

```
mongod --configsvr --replSet conf --dbpath conf2 --port 4002
```
Command shell

Process the following command in a new Terminal window

```
mongo -port 4001
rs.initiate()
rs.conf()
rs.add("localhost:4002")
rs.status()
exit
```
port 4001

```
mongo --port 4002
rs.slaveOk()
exit
```
port 4002

Experiment

Process the following command in a new Terminal window to create data shards (two replication sets each one consisting of two servers)

Command shell
```
mkdir data1-1
mkdir data1-2
```

Process the following command in a new Terminal window

Command shell
```
mongod --shardsvr --replSet data1 --dbpath data1-1 --port 4003
```

Process the following command in a new Terminal window

Command shell
```
mongod --shardsvr --replSet data1 --dbpath data1-2 --port 4004
```

Experiment

Process the following command in a new Terminal window

```
mongo -port 4003
rs.initiate()
rs.conf()
rs.add("localhost:4004")
rs.status()
exit
```

port 4003

```
mongo --port 4004
rs.slaveOk()
exit
```

port 4003

Process the following command in a new Terminal window

```
mkdir data2-1
mkdir data2-2
```

Command shell

Process the following command in a new Terminal window

```
mongod --shardsvr --replSet data2 --dbpath data2-1 --port 4005
```

Command shell

Experiment

Process the following command in a new Terminal window

```
mongod --shardsvr --replSet data2 --dbpath data2-2 --port 4006
```
Command shell

Process the following command in a new Terminal window

```
mongo -port 4005
rs.initiate()
rs.conf()
rs.add("localhost:4006")
rs.status()
exit
```
port 4005

```
mongo --port 4006
rs.slaveOk()
exit
```
port 4005

Experiment

Process the following command in a new Terminal window to start mongos

Command shell
```
mongos --configdb conf/localhost:4001,localhost:4002 --port 4000
```

Process the following command in a new Terminal window to create shards on replica sets on ports 4003 and 4005

Command shell
```
mongo --port 4000
sh.addShard("data1/localhost:4003")
sh.addShard("data2/localhost:4005")
```

List the data shards

getSiblingDB()
```
db.getSiblingDB("config").shards.find()
```

Experiment

Enable sharding of a database

```
sh.enableSharding("test")
db.getSiblingDB("config").databases.find()
```

enableSharding()

Insert a collection into shard

```
use test
db.testcol.insert({"fname":"James", "lname":"Bond", "shard-key":"000"})
db.testcol.insert({"fname":"James", "lname":"Bond", "shard-key":"001"})
db.testcol.insert({"fname":"James", "lname":"Bond", "shard-key":"002"})
db.testcol.insert({"fname":"James", "lname":"Bond", "shard-key":"003"})
db.testcol.insert({"fname":"James", "lname":"Bond", "shard-key":"004"})
db.testcol.insert({"fname":"James", "lname":"Bond", "shard-key":"005"})
db.testcol.insert({"fname":"James", "lname":"Bond", "shard-key":"006"})
db.testcol.insert({"fname":"James", "lname":"Bond", "shard-key":"007"})
```

insert()

Create index on `"shard-key"`

```
db.testcol.createIndex({"shard-key":1})
```

createIndex()

Experiment

Shard a collection

```
sh.shardCollection("test.testcol", {"shard-key": 1})
```
shardCollection()

Get statistics

```
db.test_collection.stats()
```
stats()

```
...    ...   ...   ...
"shards" : {
   "data2" : {
              "ns" : "test.testcol",
              "size" : 592,
              "count" : 8,
              "avgObjSize" : 74,
              "storageSize" : 32768,
...    ...   ...   ...
```
Statistics

Experiment

Create large collection "test_collection"

```
test_collection
use test
var bulk = db.test_collection.initializeUnorderedBulkOp();
people = ["Marc", "Bill", "George", "Eliot", "Matt", "Trey", "Tracy",
"Greg", "Steve", "Kristina", "Katie", "Jeff"];
for(var i=0; i<1000000; i++){
    user_id = i;
    name = people[Math.floor(Math.random()*people.length)];
    number = Math.floor(Math.random()*10001);
    bulk.insert( { "user_id":user_id, "name":name, "number":number });
}
bulk.execute();
```

Create index on "user_id"

```
createIndex()
db.test_collection.createIndex({user_id:1})
```

Experiment

Shard a colection over "user_id"

shardCollection()

```
sh.shardCollection("test.test_collection",{"user_id":1})
```

Get statistics

shardCollection()

```
sh.shardCollection("test.test_collection",{"user_id":1})
```

Statistics

```
...    ...   ...   ...
"shards" : {
        "data1" : {
            "ns" : "test.test_collection",
            "size" : 184478484,
            "count" : 2604399,
            "avgObjSize" : 70,
            "storageSize" : 80326656,
...    ...   ...   ...
```

Statistics

```
...    ...   ...   ...
        "data2" : {
            "ns" : "test.test_collection",
            "size" : 135758751,
            "count" : 1916602,
            "avgObjSize" : 70,
            "storageSize" : 67858432,
...    ...   ...   ...
```

Experiment

Find sharding status

```
sh.status()
```

status()

```
databases:
{  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                data1    1
                        { "_id" : { "$minKey" : 1 } } -->>
                        { "_id" : { "$maxKey" : 1 } } on : data1 Timestamp(1, 0)
...    ...    ...    ...
```

Status

Experiment

Sharding status

```
Status
test.test_collection
        shard key: { "user_id" : 1 }
        unique: false
        balancing: true
        chunks:
                data1   10
                data2   9
        { "user_id" : { "$minKey" : 1 } } -->> { "user_id" : 0 } on : data2 Timestamp(8, 0)
        { "user_id" : 0 } -->> { "user_id" : 166667 } on : data2 Timestamp(9, 0)
        { "user_id" : 166667 } -->> { "user_id" : 289501 } on : data1 Timestamp(9, 1)
        { "user_id" : 289501 } -->> { "user_id" : 414501 } on : data1 Timestamp(7, 5)
        { "user_id" : 414501 } -->> { "user_id" : 479349 } on : data1 Timestamp(7, 6)
        { "user_id" : 479349 } -->> { "user_id" : 604349 } on : data1 Timestamp(3, 0)
        { "user_id" : 604349 } -->> { "user_id" : 789699 } on : data1 Timestamp(5, 0)
        { "user_id" : 789699 } -->> { "user_id" : 958699 } on : data2 Timestamp(7, 1)
        { "user_id" : 958699 } -->> { "user_id" : 1167398 } on : data2 Timestamp(3, 2)
        { "user_id" : 1167398 } -->> { "user_id" : 1417399 } on : data2 Timestamp(3, 3)
        { "user_id" : 1417399 } -->> { "user_id" : 1667400 } on : data2 Timestamp(3, 4)
        { "user_id" : 1667400 } -->> { "user_id" : 1982999 } on : data2 Timestamp(3, 5)
        { "user_id" : 1982999 } -->> { "user_id" : 2232999 } on : data1 Timestamp(5, 2)
        { "user_id" : 2232999 } -->> { "user_id" : 2483000 } on : data1 Timestamp(5, 3)
        { "user_id" : 2483000 } -->> { "user_id" : 2733001 } on : data1 Timestamp(5, 4)
        { "user_id" : 2733001 } -->> { "user_id" : 3007999 } on : data1 Timestamp(5, 5)
        { "user_id" : 3007999 } -->> { "user_id" : 3257999 } on : data2 Timestamp(6, 2)
        { "user_id" : 3257999 } -->> { "user_id" : 3520999 } on : data2 Timestamp(6, 3)
        { "user_id" : 3520999 } -->> { "user_id" : { "$maxKey" : 1 } } on : data1 Timestamp(7, 0)
```

Experiment

Sharding status

```
Status
                test.testcol
                        shard key: { "shard-key" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                data2   1
                        { "shard-key" : { "$minKey" : 1 } } -->>
                        { "shard-key" : { "$maxKey" : 1 } } on : data2 Timestamp(1, 0)
```

Sharding

Outline

- Basics
- Why sharding ?
- Architecture
- Distribution
- Experiment
- Querying and indexing
- Production

Querying and indexing

From a database application perspective, there is no difference between querying a sharded cluster and querying a single unsharded database

In both cases, the query interface and the process of iterating over the result set are the same

How does it work inside the system ?

Config servers maintain a mapping of shard key ranges to shards

If a query includes a shard key, then mongos can quickly find which shard contains the query result set; it is called a targeted query

If the shard key is not part of the query then a query planner visits all shards to fulfill the query completely; it is called as a global or scatter/gather query

In sharded environment indexing is an important part of optimizing performance

Querying and indexing

Indexing of a sharded cluster has the following properties:

- each shard maintains its own indexes
- when an index is created on a sharded collection, each shard builds a separate index for its portion of the collection
- it means that the sharded collections on each shard should have the same indexes, otherwise query performance is inconsistent
- sharded collections permit unique indexes on the `"_id"` field and on the shard key only
- unique indexes are prohibited elsewhere because enforcing them would require inter-shard communication

Sharding

Outline

- Basics
- Why sharding ?
- Architecture
- Distribution
- Experiment
- Querying and indexing
- Production

Production

Theoretically to launch the sample MongoDB shard cluster, you had to start a total of nine processes (three mongods for each replica set, plus three config servers

In practice there is no need for so many processes

Replicated mongods are the most resource-intensive processes in a shard cluster and must be given their own machines

Replica set arbiters incur little overhead and they don't need their own servers.

Config servers store a relatively small amount of data

This means that config servers don't necessarily need their own machines, either

Production

From what you already know about replica sets and shard clusters, you can construct a list of minimum deployment requirements:

- each member of a replica set, whether it's a complete replica or an arbiter, needs to live on a distinct machine
- every replicating replica set member needs its own machine.
- replica set arbiters are lightweight enough to share a machine with another process
- config servers can optionally share a machine; the only hard requirement is that all config servers in the config cluster reside on distinct machines

References

Banker K., Bakkum P., Verch S., Garret D., Hawkins T., MongoDB in Action, 2nd ed., Manning Publishers, 2016

MongoDB Manual, Sharding