IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-3, Issue-9 E-ISSN: 2347-2693

Parallel Aggregation on Sharded Clusters

T. Mothilal[1*] and P. Anil Kumar[2]

[1*,2]Department of Computer Science and Engineering,
Prasad V Potluri Siddhartha Institute of Technology, Kanuru, India

www.ijcseonline.org

Received: Aug/12/2015 Revised: Aug/20/2015 Accepted: Sep/17/2015 Published: Sep/30/2015

***Abstract*—** The data is organized by databases with the help of database management systems. DBMS is the collection of schemas, queries and other objects. To aggregate the data DBMS used Cartesian products between two or more tables and produce a result in a logical table. Where data is increasing rapidly day by day, so writing joins on large tables is difficult to data analysts and manage complex queries on large scale table is quite difficult to DBMS. To reduce complexity of manipulating large data schemaless databases are introduced. MongoDB process schemaless data and having more use cases to achieve parallel processing on data. Aggregation is one of the function which is applying on the data. To get fastest aggregation results use mongodb sharded cluster and mareduce.

***Keywords*—** NOSQL, MongoDB, Sharding, Parallelism, MapReduce

I. INTRODUCTION

Database is typically organizing collection of data. A DBMS is a software application that is used to organize the databases. Generally DBMS is designed to allow the definition, creation, querying and administrating the data in databases. DBMS are classified according to the database model. In early days database technology has developed in different eras, in 1960's as navigational model, in 1970's as relational model, in 1990's as Object model and in 2000's as NoSQL model. In the navigational database model, the data is processed in sequential order by an interactive manner and later on B-trees are added. In the traditional relational database model, the data is stored with fixed tables of different entities. Here we can link the tables based the key's i.e., data fields of the table. In the Object data models the data is stored in the respective databases as objects. So the relation between data is allowed in the form of relation between objects. In the NoSQL databases the data is stored in key-value pairs in the document oriented based approach [3]. These databases are very fast, avoids join operations by storing the data as de-normalized data. It avoids fixed table schemas and horizontally scalable.

II. OVERVIEW

MongoDB is a document based database, not a relational one. The row in RDBMS is considered as single document mongodb. Table in RDBMS is considered as collection in mongodb [4]. A collection can have many documents. Each document has set of key/value pairs. Mongodb generates a unique id for every document which can works as default primary key. Here key/value pairs are independent that means there are no need to be same all the key/value pairs in a single document. So the document can have variety of data objects like a text, numerical, binary, and so on. Mongodb is also easy to scaling because a collection can dynamically increase the documents and document can also update with a new key/value pairs. The seeking time of a data is very less in mongodb when compare RDBMS. It is the beauty of mongodb, because in RDBMS the processing of data done in sequential manner. Where as in mongodb supports for indexing, so data can found directly at that particular index. Mongodb can access large data most efficient than RDBMS [5].

MapReduce in MongoDB:

To aggregate huge data mapreduce is most useful in mongodb. Mongodb mapreduce uses the javascript as its query language. Mapreduce can process across multiple servers parallel. Mongodb can also apply mapreduce on sharded cluster to achieve more efficiency. The mapreduce has two phases like map function and reduce function. There is no need to wait completion of map function, reduce function is also execute parallel. The mapreduce instances are created and each instance is works on different shards [2]. How many shards are present in cluster that many instances are creates. Each shard is having part of data. The data may store as single chunk or multiple chunks. Each shard produce different results for map function, reduce

function will combine all the results of map function as required output.

Aggregation using mapreduce:

Aggregation will perform mathematical operations like sum, count, average, etc., Applying aggregating function on large data in mogodb is more efficient with help of mapreduce. Aggregate function can be done match and group operations on the data [1]. The match function will select the key/values based on the condition and group function will combine all the values with a corresponding key. The mapreduce works with an aggregate framework by applying query on the data and results obtained from the query, apply the map function and get the results with the grouped key/value pairs on each shard. Reduce function will combine all the results from each shard and produce a required output based on the function.

```
db.books.mapReduce(function()
        {
        var category;
        if ( this.pages >= 250 )
        category = 'Big Books';
        else
        category = "Small Books";
        emit(category, {name: this.name});
        },
function(key, values)
        {
        var sum = 0;
        values.forEach(function(doc) {
        sum += 1;
        }
        return {books: sum};
        },
        {
out:"book_results"
});
```

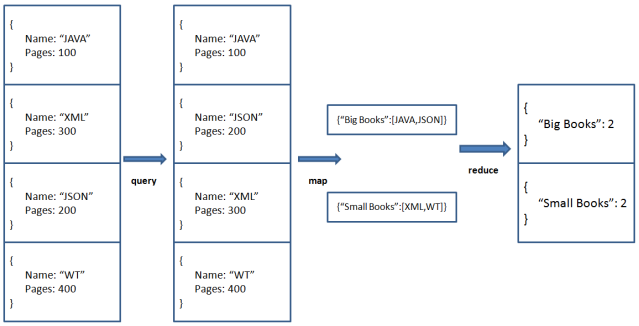


Fig.1. Aggregation using Mapreduce

When applying mapreduce on books collections, group the collection based on the query and produce map results. Reduce function can perform count the number of books from map results.

Sharded cluster:

Processing of data on multiple systems in mongodb is nothing but a sharded cluster. Shards are created by developer. Data can be stored in shards using shard key. Each shard is independent. There is no redundancy of data one shard and another shard. Each shard is having primary and secondary. Mapreduce job running on a shard, the secondary shard is acts as primary when primary shard is failed [7]. Mongodb provides high availability of data with sharded cluster, because the data is replicating in secondary also. To deploy sharded cluster we need to bellow components:

a. **Shards:** shards are used to store the data. Each shard is having one primary node and two or more secondary nodes.
b. **Config servers:** config servers are having clusters information. The query router periodically communicates with the config server to get the information of the cluster.
c. **Query routers:** Query Routers are basically mongos instances, interface with client applications and direct operations to the appropriate shard. The query routers process the data and produce results.

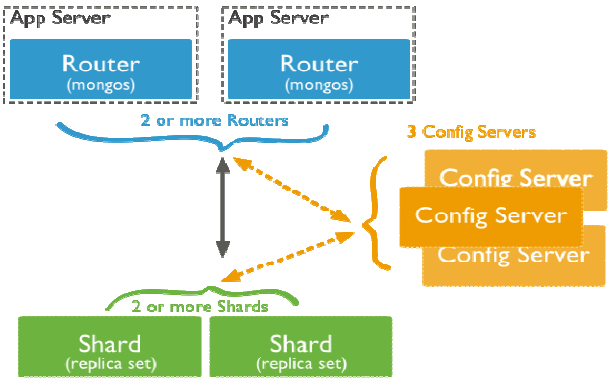


Fig.2. components of a sharded cluster

III. IMPLEMENTATION

a. Installation of MongoDB on ubuntu

- Import the public key used by the package management system.
  huser@incu-nn2$sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
- Create a list file for MongoDB
  huser@incu-nn2$echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee /etc/apt/sources.list.d/mongodb.list
- Reload local package database
  huser@incu-nn2$sudo apt-get update
- Install the MongoDB packages.
  huser@incu-nn2$sudo apt-get install mongodb-org
- Start MongoDB
  huser@incu-nn2$sudo /etc/init.d/mongod start
  (or)
  huser@incu-nn2$sudo service mongod start
- Verify that MongoDB has started successfully /var/log/mongodb/mongod.log.
- Stop MongoDB
  huser@incu-nn2$sudo /etc/init.d/mongod stop
  (or)
  huser@incu-nn2$sudo service mongod stop
- Restart MongoDB
  huser@incu-nn2$sudo /etc/init.d/mongod restart
  (or)
  huser@incu-nn2$sudo service mongod restart
- Begin using MongoDB
  huser@incu-nn2$mongo

b. Deploy sharded cluster

Step 1: Start three shard servers i.e., mongod data servers. Create data directories for shard servers.

➢ Start Replicaset "a"

```
$mkdir a0
$mkdir a1
$mkdir a2
$mongod --shardsvr --replSet a --dbpath a0 --logpath log.a0
--port 27000 –fork --logappend
$mongod --shardsvr --replSet a --dbpath a1 --logpath log.a1
--port 27001 --fork –logappend
$mongod --shardsvr --replSet a --dbpath a2 --logpath log.a2
--port 27002 --fork –logappend
```

➢ Start Replicaset "b"

```
$mkdir b0
$mkdir b1
$mkdir b2
$mongod --shardsvr --replSet b --dbpath b0 --logpath log.b0
--port 27100 --fork –logappend
$mongod --shardsvr --replSet b --dbpath b1 --logpath log.b1
--port 27101 --fork –logappend
$mongod --shardsvr --replSet b --dbpath b2 --logpath log.b2
--port 27102 --fork –logappend
```

➢ Start Replicaset "c"

```
$mkdir c0
$mkdir c1
$mkdir c2
$mongod --shardsvr --replSet c --dbpath c0 --logpath log.c0
--port 27200 --fork –logappend
$mongod --shardsvr --replSet c --dbpath c1 --logpath log.c1
--port 27201 --fork –logappend
$mongod --shardsvr --replSet c --dbpath c2 --logpath log.c2
--port 27202 --fork –logappend
```

➢ Start Replicaset "d"

```
$mkdir d0
$mkdir d1
$mkdir d2
$mongod --shardsvr --replSet d --dbpath d0 --logpath log.d0
--port 27300 --fork –logappend
$mongod --shardsvr --replSet d --dbpath d1 --logpath log.d1
--port 27301 --fork –logappend
$mongod --shardsvr --replSet d --dbpath d2 --logpath log.d2
--port 27302 --fork –logappend
```

Step 2: Start the Config Server Database Instances. config servers are having clusters information. The query router periodically communicates with the config server to get the information of the cluster.

```
$sudo mkdir /configdb0
```

```
$sudo mkdir /configdb1
$sudo mkdir /configdb2
$mongod --configsvr --dbpath configdb0 --port 27020 --fork --logpath log.cfg0 --logappend
$mongod --configsvr --dbpath configdb1 --port 27021 --fork --logpath log.cfg1 --logappend
$mongod --configsvr --dbpath configdb2 --port 27022 --fork --logpath log.cfg2 –logappend
```

Step 3: Start the mongos instances. The mongos instances are lightweight and do not require data directories. By default, a mongos instance runs on port 27017.

huser@incu-nn2$mongos --configdb incu-nn2:27020, incu-nn2:27021, incu-nn2:27022 --fork --logappend --logpath log.mongos0

Step 4: Initiate Replicaset “a” for shard. Here 27000 is primary instance and 27001, 27002 are secondary instances.

huser@incu-nn2$mongo --port 27000

```
>rs.initiate()
>rs.add(“incu-nn2:27001”)
>rs.add(“incu-nn2:27002”)
>rs.status()
```

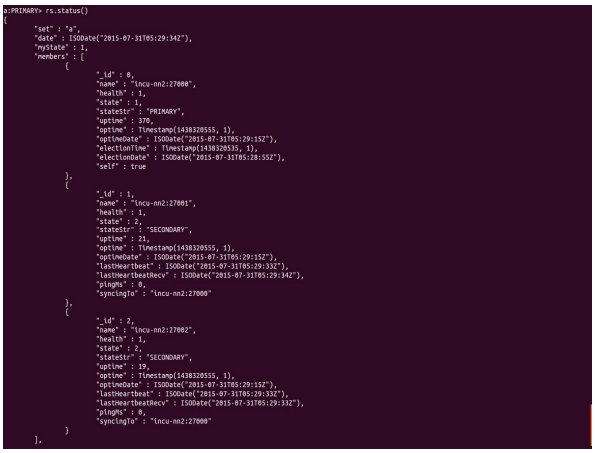


Fig.3. Sharded ReplicaSet “a” Status

Step 5: Initiate Replicaset “b” for shard. Here 27100 is primary instance and 27101, 27102 are secondary instances.

huser@incu-nn2$mongo --port 27100

```
>rs.initiate()
>rs.add(“incu-nn2:27101”)
>rs.add(“incu-nn2:27102”)
>rs.status()
```

Step 6: Initiate Replicaset “c” for shard. Here 27200 is primary instance and 27201, 27202 are secondary instances.

huser@incu-nn2$mongo --port 27200

```
>rs.initiate()
>rs.add(“incu-nn2:27201”)
>rs.add(“incu-nn2:27202”)
>rs.status()
```

Step 7: Initiate Replicaset “d” for shard. Here 27300 is primary instance and 27301, 27302 are secondary instances.

Huser@incu-nn2$mongo --port 27300

```
>rs.initiate()
>rs.add(“incu-nn2:27301”)
>rs.add(“incu-nn2:27302”)
>rs.status()
```

Step 8: Add shards to the cluster on mongos instance.

```
$mongo
>sh.addShard(“a/incu-nn2:27000”)
>sh.addShard(“b/incu-nn2:27100”)
>sh.addShard(“c/incu-nn2:27200”)
>sh.addShard(“d/incu-nn2:27300”)
>sh.status()
```

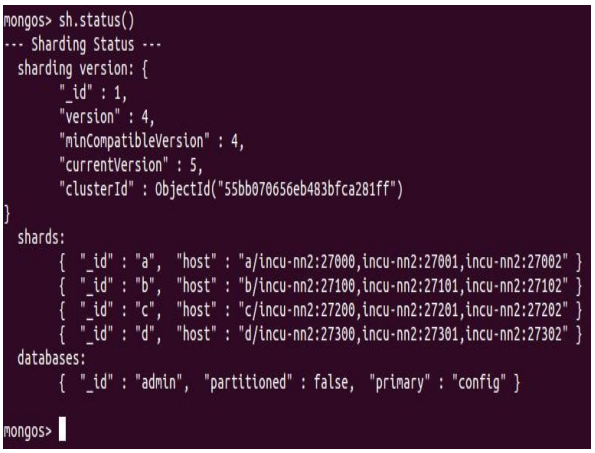
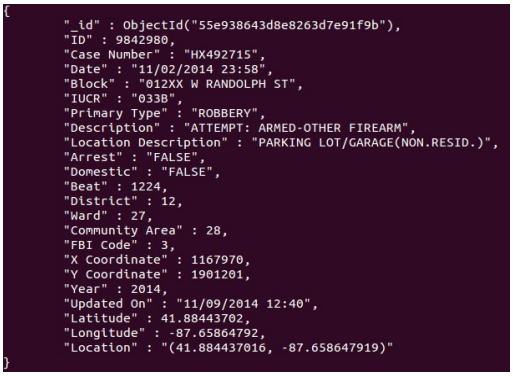


Fig.4. Sharding Status

IV. RESULTS

Consider the collection having the data that is crimes occurred in Chicago in the period of 2012 to 2014. And there is having 1048575 documents. When applying mapreduce on that collection for number of crimes occurred per day without sharding it will takes 21784 milli seconds.

When applying mapreduce on that collection for number of crimes occurred per day with sharding it will takes 10542 milli seconds.

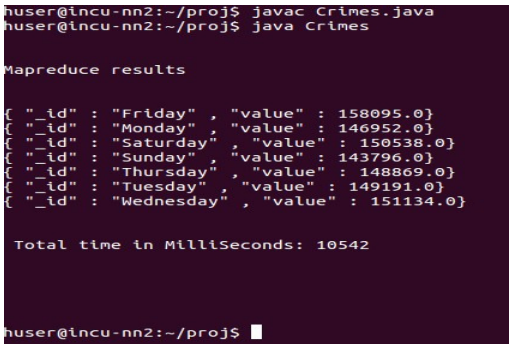


```
{
        "_id" : ObjectId("55e938643d8e8263d7e91f9b"),
        "ID" : 9842980,
        "Case Number" : "HX492715",
        "Date" : "11/02/2014 23:58",
        "Block" : "012XX W RANDOLPH ST",
        "IUCR" : "033B",
        "Primary Type" : "ROBBERY",
        "Description" : "ATTEMPT: ARMED-OTHER FIREARM",
        "Location Description" : "PARKING LOT/GARAGE(NON.RESID.)",
        "Arrest" : "FALSE",
        "Domestic" : "FALSE",
        "Beat" : 1224,
        "District" : 12,
        "Ward" : 27,
        "Community Area" : 28,
        "FBI Code" : 3,
        "X Coordinate" : 1167970,
        "Y Coordinate" : 1901201,
        "Year" : 2014,
        "Updated On" : "11/09/2014 12:40",
        "Latitude" : 41.88443702,
        "Longitude" : -87.65864792,
        "Location" : "(41.884437016, -87.658647919)"
}
```


Fig.5. Crime dataset

```
huser@incu-nn2:~/proj$ javac Crimes.java
huser@incu-nn2:~/proj$ java Crimes

Mapreduce results

{ "_id" : "Friday" , "value" : 158095.0}
{ "_id" : "Monday" , "value" : 146952.0}
{ "_id" : "Saturday" , "value" : 150538.0}
{ "_id" : "Sunday" , "value" : 143796.0}
{ "_id" : "Thursday" , "value" : 148869.0}
{ "_id" : "Tuesday" , "value" : 149191.0}
{ "_id" : "Wednesday" , "value" : 151134.0}

 Total time in MilliSeconds: 21784

huser@incu-nn2:~/proj$
```


Fig.6. MapReduce on MongoDB without sharding



```
huser@incu-nn2:~/proj$ javac Crimes.java
huser@incu-nn2:~/proj$ java Crimes

Mapreduce results

{ "_id" : "Friday" , "value" : 158095.0}
{ "_id" : "Monday" , "value" : 146952.0}
{ "_id" : "Saturday" , "value" : 150538.0}
{ "_id" : "Sunday" , "value" : 143796.0}
{ "_id" : "Thursday" , "value" : 148869.0}
{ "_id" : "Tuesday" , "value" : 149191.0}
{ "_id" : "Wednesday" , "value" : 151134.0}

 Total time in MilliSeconds: 10542

huser@incu-nn2:~/proj$
```


Fig.7. MapReduce on MongoDB Shards

V. CONCLUSION

Processing large datasets having 1048575 records is difficult or time taking process in normal systems. Using MongoDB Schemaless databases users can achieve better performance than traditional systems. MongoDB without distribution, processing same dataset is taken much time than mongodb with sharded distributed cluster. So, distributed processing using mongodb is efficient and highly useful to bigdata processing.

REFERENCES

[1] J. M. Hellerstein, "The case for online aggregation", Technical Report UCB//CSD-96-908, EECS Computer Science Division, University of California, Berkeley, CA,**1996**

[2] Jeffrey Dean and Sanjay Ghemawat "MapReduce: Simplified Data Processing on Large Clusters", OSDI **2014**

[3] Anju abraha,"A Dynamic Query Form System for Mongodb", SSRG-IJCSE, volume-1 issue-9, Nov **2014**.

[4] MongoDB, "http://docs.mongodb.org/manual/", Thursday, April 30, **2015**.

[5] MongoDB, http://www.tutorialspoint.com/mongodb/, Monday, July 6, **2015**

[6] Replication, http://stackoverflow.com, Tuesday, August 11, **2015**

[7] Sharding, http://gist.github.com, Monday, August 17, **2015**.

AUTHORS PROFILE

T.Mothilal is presently M.Tech Student, Dept. of Computer Science & Engineering, Prasad V Potluri Siddhartha Institute of Technology (Autonomous), kanuru, India.

P. Anil Kumar is presently Assistant Professor, Dept. of Computer Science & Engineering, Prasad V Potluri Siddhartha Institute of Technology (Autonomous), kanuru, India.