Z604 - Homework2 (nishshah)

**Data Introduction**

All information about data used can be found at this link:

http://stat-computing.org/dataexpo/2009/the-data.html

I have used data of year 2008.

**Data loading into mongo through command:**

mongoimport --db z604 --collection flight --type csv --headerline --file /home/ubuntu/2008.csv

Go to Mongo shell using command: mongo

Use database and collection which were created from mongoimport command: use z604

**1.) DB statistics and Indexing.**

**DB statistics right after loading data into Mongo.**

{

"db" : "z604",

"collections" : 1,

"objects" : 7009728,

"avgObjSize" : 498.1848970459339,

"dataSize" : 3492140622,

"storageSize" : 902234112,

"numExtents" : 0,

"indexes" : 1,

"indexSize" : 63619072,

"ok" : 1

}

**Default index created by Mongo:**

db.flight.getIndexes()

[

{

"v" : 1,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "z604.flight"

}

]

**However the search performance will increase if we can create index for FlightNumber, Origin, Destination.**

Commands for that:

db.flight.createIndex({"FlightNum":1})

db.flight.createIndex({"Origin":1})

db.flight.createIndex({"FlightNum":1,"Origin":1})

db.flight.createIndex({"Dest":1})

db.flight.getIndexes()

[

{

"v" : 1,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "z604.flight"

},

{

"v" : 1,

"key" : {

"FlightNum" : 1

},

"name" : "FlightNum\_1",

"ns" : "z604.flight"

},

{

"v" : 1,

"key" : {

"Origin" : 1

},

"name" : "Origin\_1",

"ns" : "z604.flight"

},

{

"v" : 1,

"key" : {

"FlightNum" : 1,

"Origin" : 1

},

"name" : "FlightNum\_1\_Origin\_1",

"ns" : "z604.flight"

},

{

"v" : 1,

"key" : {

"Dest" : 1

},

"name" : "Dest\_1",

"ns" : "z604.flight"

}

]

**DB statistics after indexing.**

db.stats()

{

"db" : "z604",

"collections" : 2,

"objects" : 7115476,

"avgObjSize" : 492.9593674407728,

"dataSize" : 3507640548,

"storageSize" : 904335360,

"numExtents" : 0,

"indexes" : 6,

"indexSize" : 192880640,

"ok" : 1

}

**Here in the following questions, we do not require to use Dest field, so we can ignore creating index on that field, hence it will occupy less memory and performance will be the same.**

**Some performance comparision of find() query before and after indexing.**

**Query:db.flight.find({FlightNum:1467}).explain("executionStats")**

**Before Indexing:**

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 2385,

**"executionTimeMillis" : 3935,**

"totalKeysExamined" : 0,

"totalDocsExamined" : 7009728,

"executionStages" : {

"stage" : "COLLSCAN",

"filter" : {

"FlightNum" : {

"$eq" : 1467

}

},

"nReturned" : 2385,

"executionTimeMillisEstimate" : 3640,

"works" : 7009730,

"advanced" : 2385,

"needTime" : 7007344,

"needYield" : 0,

"saveState" : 54763,

"restoreState" : 54763,

"isEOF" : 1,

"invalidates" : 0,

"direction" : "forward",

"docsExamined" : 7009728

}

}

**After Indexing:**

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 2385,

**"executionTimeMillis" : 9,**

"totalKeysExamined" : 2385,

"totalDocsExamined" : 2385,

"executionStages" : {

"stage" : "FETCH",

"nReturned" : 2385,

"executionTimeMillisEstimate" : 10,

"works" : 2386,

"advanced" : 2385,

"needTime" : 0,

"needYield" : 0,

"saveState" : 18,

"restoreState" : 18,

"isEOF" : 1,

"invalidates" : 0,

"docsExamined" : 2385,

"alreadyHasObj" : 0,

"inputStage" : {

"stage" : "IXSCAN",

"nReturned" : 2385,

"executionTimeMillisEstimate" : 10,

"works" : 2386,

"advanced" : 2385,

"needTime" : 0,

"needYield" : 0,

"saveState" : 18,

"restoreState" : 18,

"isEOF" : 1,

"invalidates" : 0,

"keyPattern" : {

"FlightNum" : 1

},

"indexName" : "FlightNum\_1",

"isMultiKey" : false,

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 1,

"direction" : "forward",

"indexBounds" : {

"FlightNum" : [

"[1467.0, 1467.0]"

]

},

"keysExamined" : 2385,

"dupsTested" : 0,

"dupsDropped" : 0,

"seenInvalidated" : 0

}

}

},

**You can clearly see there is a significance difference between execution times of both queries.**

**So Indexing is always a good strategy when there are too many search query to database but it will be waste of space if we create indexes for least used fields in query.**

**2.) Aggregate and MapReduce strategy for finding total delays.**

**Aggregate query:**

db.flight.aggregate([{$group: {

\_id: {flightNum:"$FlightNum",Origin:"$Origin"},

total\_WeatherDelay:{$sum:"$WeatherDelay"},

total\_CarrierDelay:{$sum:"$CarrierDelay"},

total\_SecurityDelay:{$sum:"$SecurityDelay"}}

},

{$group: {\_id: null,count: { $sum: 1 }}}

])

**Result: { "\_id" : null, "count" : 105748 }**

**We can remove second $group object ro see the actual results rather than just count.**

**MapReduce query:**

var mapFunction1 = function() {

var value = {

total\_WeatherDelay: this.WeatherDelay,

total\_CarrierDelay:this.CarrierDelay,

total\_SecurityDelay:this.SecurityDelay

};

emit({flightNum: this.FlightNum,Origin:this.Origin}, value);

};

var reduceFunction1 = function(key,ObjVals) {

reducedVal = { total\_SecurityDelay: 0,total\_WeatherDelay:0,total\_CarrierDelay:0 };

for (var idx = 0; idx < ObjVals.length; idx++) {

reducedVal.total\_WeatherDelay+=ObjVals.total\_WeatherDelay;

reducedVal.total\_CarrierDelay+=ObjVals.total\_CarrierDelay;

reducedVal.total\_SecurityDelay+=ObjVals.total\_SecurityDelay

}

return reducedVal;

};

db.flight.mapReduce(mapFunction1,reduceFunction1,{ out: "map\_reduce\_example"})

**Result:**

{

"result" : "map\_reduce\_example",

"timeMillis" : 127585,

"counts" : {

"input" : 7009728,

"emit" : 7009728,

"reduce" : 1517114,

"output" : 105748

},

"ok" : 1

}

**We can see that the question can be solved by both aggregate() and mapReduce() but execution time for both the queries differs a lot.**

**mapReduce takes more time than aggregate because of two operations, map and reduce take too much of computation. (Both, with and without indexes).**

**aggregate() provides inline results whereas mapReduce() creates a new collection and store the result in it.**

**In both the queries, results are grouped by FlightNum and Origin, i.e. they work as composite object/key. Values are also composite objects.**

**Below are sample queries where both key and value are single object**.

**Aggregate query:**

db.flight.aggregate([{$group: {

\_id: "$FlightNum",

total\_WeatherDelay:{$sum:"$WeatherDelay"}}

},

{$group: {\_id: null,count: { $sum: 1 }}}

])

**MapReduce query:**

var mapFunction = function() {

emit(this.FlightNum, this.WeatherDelay);

};

var reduceFunction = function(key,value) {

return Array.sum(value);

};

db.flight.mapReduce(mapFunction,reduceFunction,{ out: "map\_reduce\_example"})

**These queries will give total delay due to weather for each FlightNumber.**

**3.) MapReduce task for the data.**

**There can be many mapReduce tasks defined for this dataset. E.g. keys could be UniqueCarrier, FlightNum, Origin, Dest,Month,DayOfWeek and combinations of those.**

**Values could be Total Distance travelled, how many times the flight has been cancelled/diverted, Avg. Taxi Out/Taxi In for each origin and destination respectively, Various total delays for each Carrier/origin/destination/FlightNum/Month.**

**Queries:**

1)MapReduce query:

var mapFunction = function() {

emit(this.UniqueCarrier, this.Cancelled);

};

var reduceFunction = function(key,value) {

return Array.sum(value);

};

db.flight.mapReduce(mapFunction,reduceFunction,{ out: "**carrier\_cancellation**"})

2)MapReduce query:

var mapFunction = function() {

emit(this.FlightNum, this.Distance);

};

var reduceFunction = function(key,value) {

return Array.sum(value);

};

db.flight.mapReduce(mapFunction,reduceFunction,{ out: "**flight\_distance\_travelled**"})

3)MapReduce query:

var mapFunction = function() {

var value = {

total\_ArrDelay: this.ArrDelay,

total\_DepDelay:this.DepDelay

};

emit(this.Month, value);

};

var reduceFunction = function(key,value) {

var reducedVal = 0;

reducedVal += value.total\_ArrDelay+value.total\_DepDelay

return reducedVal;

};

db.flight.mapReduce(mapFunction,reduceFunction,{ out: "t**otal\_delay\_monthly**"})