Implementasi Dan Analisis Performansi Mapreduce di Lingkungan Sistem Basisdata Berbasis Dokumen Terdistribusi Homogen

Implementation and Analysis of MapReduce's Performance in a Homogeneous Distributed Document-Oriented Database Environment

Hegar Aryo Dewandaru[1], Kemas Rahmat Saleh W., S.T., M.Eng.[2], Alfian Akbar Gozali, S.T., M.T.[3]
[1, 2, 3] Program Studi S1 Teknik Informatika, Fakultas Informatika, Universitas Telkom
[1]hegararyo@hotmail.co.id, [2]bagindokemas@telkomuniversity.ac.id, [3]alfian@tass.telkomuniversity.ac.id

Abstrak - Pada sebuah perusahaan besar, sangatlah penting untuk memiliki manajemen sistem basisdata yang mampu menampung seluruh data dan dokumen milik karyawan. Data tersebut akan sangat besar sehingga tidak akan memunkinkan untuk ditampung oleh *single server*. Untuk menyelesaikan masalah tersebut, data yang sangat besar itu dapat didistribusikan ke dalam beberapa *cluster*. Di lingkungan terdistribusi inilah implementasi dari metode MapReduce akan sangat bermanfaat bagi sistem. MapReduce adalah sebuah operasi untuk menyelesaikan masalah yang mirip dengan algoritma *divide and conquer*. Sesuai dengan namanya, MapReduce terdiri dari proses *map* (pemetaan) suatu data dan *reduce* (pengurangan) yang berakhir pada penggabungan data-data yang sama.

Pada tugas akhir ini akan dilakukan penelitian terhadap performansi MapReduce di lingkungan sistem basisdata terdistribusi homogen. Untuk membangun lingkungan tersebut, sebelumnya harus dilakukan pengecekan terhadap setiap komputer yang digunakan. Pastikan bahwa seluruh komputer memiliki spesifikasi perangkat lunak dan keras, serta manajemen sistem basisdata yang sama. Setelah itu, dataset yang berbasis dokumen harus di-*import* ke *database* komputer yang berperan sebagai *master*. Kemudian, dengan menggunakan metode *sharding,* setiap node akan diberi peran: *master* akan berberan sebagai *router,* satu *node* sebagai *config server*, dan sisanya sebagai *shard server* sehingga terbentuklah lingkungan sistem basisdata berbasis dokumen terdistribusi homogen. Dataset kemudian akan didistribusikan ke setiap shard. Akhirnya, query MapReduce akan dijalankan dan diuji di single server dan 3 arsitektur distributed database yang berbeda untuk diteliti performansinya.

Dari hasil pengujian yang dilakukan, dapat dilihat bahwa MapReduce bekerja lebih baik di lingkungan terdistribusi dibandingkan dengan pada *single server*. Kesimpulan yang dapat diambil adalah bahwa sistem basisdata terdistribusi meningkatkan performansi MapReduce.

Kata kunci: *MapReduce*, *document-oriented database*, *distributed database system*, *distributed database management system, homogeneous distributed database*

Abstract – *In a company, it's important to have a database management system, which is capable of containing every document that is owned by their employees. The data will become very big, which makes it impossible to put the whole data into a single server. To solve this problem, that huge data can be distributed to clusters. In this distributed database environment, the implementation of MapReduce will give more contribution to the system. MapReduce is an operation, or a method that utilizes the divide-and-conquer alghorithm. As the name suggests, MapReduce consists of a mapping process and a reduce process, which will aggregate useful information from the dataset.*

In this final project, a research will be conducted to observe MapReduce's performance in a distributed database environment. First of all, it is needed to make sure that every computer that is used in the system has the exact specification of hardware, software, and database management system. After that, import the document-oriented dataset into the system's master's database. Then, by using a method called sharding, each of the computers will be given roles, so that: master as the router, one computer as a config server, and the rest as the shard servers; only after that that the homogenous distributed document-oriented database is built. The dataset will then be distributed across the shards. Finally, the MapReduce query will be executed and observed under a single server and 3 different architectures of distributed database environment.

The overall result of this research shows that MapReduce performs better in a distributed environment than in a single server. The conclusion is that the distributed database environment improves the performance of MapReduce.

Keywords: *MapReduce*, *document-oriented database*, *distributed database system*, *distributed database management system, homogeneous distributed database*

I. PENDAHULUAN

Perkembangan terkini tentang bagaimana data disimpan oleh perusahaan besar membuat kita berpikir tentang bagaimana cara memodelkan data agar lebih sesuai dengan kasus yang dihadapi. Tidak semua data tepat untuk dimodelkan menjadi tabel relasi. Untuk beberapa kasus pada perusahaan tersebut, basisdata non-relational digunakan. Tipe data yang dominan digunakan untuk bentuk basisdata non-relational adalah *document-oriented database.*

Perkembangan ilmu teknologi yang semakin cepat juga menuntut sistem penyimpanan yang lebih efektif, dimana *storage* hanya menampung data yang dibutuhkan. Namun, data tersebut tetap berukuran sangat besar sehingga tidak memungkinkan penyimpanan dengan single server.

Permasalahan tersebut dapat diselesaikan dengan menggunakan sistem basisdata terdistribusi. *Distributed Database System* (DDBS) memungkinkan pendistribusian data ke lebih dari satu server dimana setiap server merupakan sistem basisdata yang independen. Dengan begitu, mesin dapat bekerja lebih optimal dibandingkan dengan single server.

Dilihat dari arsitekturnya, ada dua jenis distributed database, yaitu *homogenous* DDBS dan *heterogenous* DDBS. Perbedaannya terletak pada spesifikasi komputer yang digunakan di setiap node pada sistemnya. Seluruh node pada *homogenous* menggunakan spesifikasi *hardware* dan *software* serta DBMS yang sama persis, sedangkan pada *heterogenous* DDBS, setiap node pada sistem boleh memiliki spesifikasi yang berbeda-beda. *Homogeneous* DDBS lebih unggul karena konfigurasinya jauh lebih mudah dibandingkan dengan *heterogeneous* DDBS yang harus membangun sistem komunikasi yang dapat menghubungkan antar dua sistem basisdata yang berbeda [1]. Penggunaan DBMS yang berbeda pada *heterogeneous* DDBS juga akan mengakibatkan query pada sistem menjadi lebih kompleks.

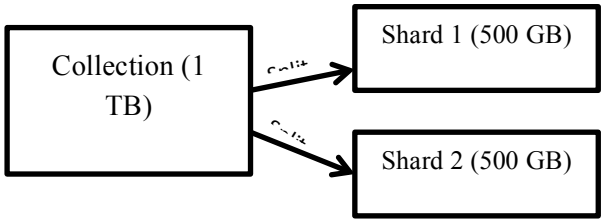
Penyimpanan data yang lebih efektif dapat diwujudkan dengan menggunakan metode mapReduce. MapReduce adalah metode yang diciptakan oleh Google yang mampu memilih dan memberikan informasi yang diinginkan oleh pengguna [2].

Dataset yang diproses memiliki baris yang cukup banyak, sehingga membutuhkan waktu yang cukup lama untuk memroses setiap instruksi atau transaksinya. Dengan kata lain, akan dibutuhkan *response time* dan *throughput* yang cukup lama jika diproses hanya dengan satu prosesor saja. Namun, ada juga kasus dimana meskipun pemrosesan data yang besar dilakukan di lebih dari satu *site* di lingkungan database terdistribusi, nilai *throughput* dan *response time* masih belum memuaskan [3].

Pada tugas akhir ini akan diimplementasikan metode MapReduce pada lingkungan *Homogenous Distributed Database* dimana seluruh node menerapkan sistem *document-oriented database.* Parameter uji yang digunakan untuk analisis performansi adalah *response time* dan *throughput.*

II. *SHARDING*

Sharding adalah metode milik MongoDB untuk meningkatkan scalability secara horizontal (Horizontal Scaling). Sharding melakukan penyimpanan data untuk lebih dari satu mesin. MongoDB menggunakan sharding untuk menyelesaikan masalah deployment dengan dataset yang besar dan operasi throughput yang tinggi dimana proses query melebihi kapasitas server tunggal [1]. Sharding akan membagi data set tersebut dan mendistribusikannya ke lebih dari satu server dimana setiap shard merpakan sebuah basisdata yang independen. Karena itulah, sharding bukan merupakan sebuah metode replikasi, melainkan distribusi.

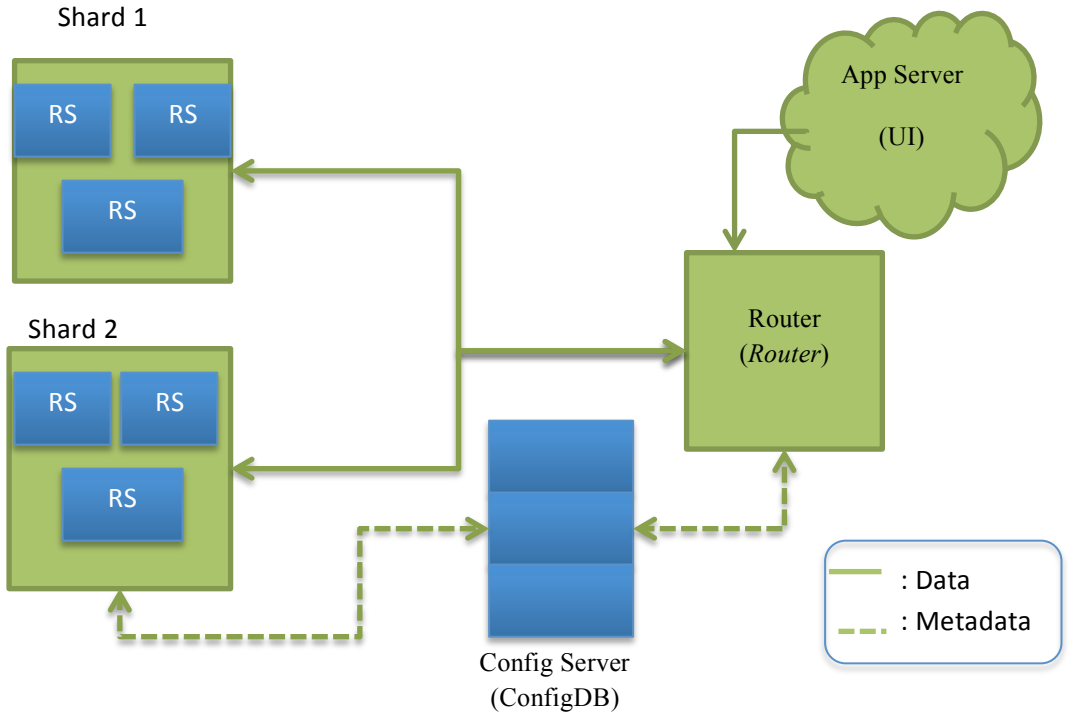


Gambar 2-1: Contoh Partisi data pada *sharding*

Setiap shard server ukurannya ditentukan secara manual ketika sistem akan dibangun. Ukuran disesuaikan dengan jumlah shard yang ada sistem dan ukuran dari dataset, sedemikian sehingga total dari ukuran shard dapat menampung seluruh data. Karena itu pada gambar 2-1, ukuran *chunks* per shard server diatur sebesar 500 GB. Metode *sharding* akan mengurangi jumlah operasi yang ditangani oleh setiap shard. Data kemudian akan didistribusikan secara merata ke setiap shard secara otomatis oleh mongoDB setelah menentukan collection mana yang akan didistribusikan.

MongoDB menjadi sistem basisdata terdistribusi dengan menggunakan *sharding,* Berikut adalah cluster-cluster yang membangun sebuah lingkungan sudah diimplementasi sharding:

1. *Query router*: Router pada sistem terdistribusi sharding. Router dapat berhubungan langsung dengan setiap *shard.* Router ini dapat diakses melalui program *mongos.exe*. MongoS memroses setiap query dan Lingkungan MongoDB yang sudah terimplementasi *sharding* juga harus memiliki minimal satu *site* yang disebut *config server*, yang bertanggung jawab dalam merekam setiap kegiatan dari sistem. Router juga bisa berfungsi sebagai client.
2. *Shards*: Tempat penyimpanan data. Sharding bisa berupa program mongod atau *replica set,* tergantung dari kebutuhan pengguna.
3. Config server: adalah tempat penyimpanan metadata dari cluster-cluster sistem. Data ini berisi *mapping* data yang akan digunakan oleh router untuk melakukan operasi terhadap cluster tersebut. Config server juga merekam setiap aktifitas yang dilakukan oleh mongos. Tanpa sebuah config server, lingkungan sudah diimplementasi sharding tidak bisa dibangun.



Gambar 2-3: Topologi Sharding [1]

Biasanya, suatu shard terdiri dari 3 *instance* replica set. Replica Set (RS) adalah strategi MongoDB untuk menangani masalah *availability* pada sistem terdistribusi MongoDB. Penggunaan replica set tidak wajib. Tanpa menggunakan Replica Set, *sharding* tetap dapat di bangun dengan menggunakan program mongoD dengan perintah --*shardsvr.*

III. MAPREDUCE

MapReduce adalah sebuah metode untuk memroses dan men-*generate* data set yang bervolume besar. Sesuai dengan namanya, MapReduce terdiri dari algoritma *map* atau pemetaan dan *reduce*, yang berarti pengurangan atau penggabungan dari data-data yang sama [2].

Setelah router meng-*input*-kan masukan berupa instruksi *document-oriented database* ke fungsi MapReduce yang dibangun di router, algoritma MapReduce akan memulai tahapan-tahapan berikut:

1. MapReduce melakukan proses mapping terhadap setiap document dengan key dan value yang ditentukan di fungsi map.
2. Pada tahap ini, seluruh value document diagregasi berdasarkan key.

3. Fungsi reduce menerima inputan, yaitu key dan juga sebuah serangkaian *value*. Kemudian, fungsi reduce akan mengelompokkan setiap dokumen berdasarkan key dan value yang merupakan output dari fungsi reduce.

Setelah seluruh tahapan ini dilalui, output dari eksekusi MapReduce akan tersedia di file output *R* yang berupa collection ke dalam database yang dilakukan sharding.

Fungsi MR (MapReduce) yang digunakan dalam penelitian ini ada 6:

3.1 Fungsi MR1

Fungsi MR1 langsung menggunakan fungsi MapReduce yang disediakan oleh MongoDB:

```
db.building.mapReduce(function()

{ emit(this.address.zipcode, this.borough) },

function(key,values) { return
```

Fungsi MapReduce ini berfungsi untuk melakukan agregasi dari *field zipcode* dan *borough.*

Return raw diberi nilai true agar memungkinkan untuk mengembalikan nilai *raw* dari collection yang di akan diterapkan fungsi mapReduce. Sedangkan *inline* adalah perintah untuk menampilkan langsung hasil dari fungsi mapReduce tanpa membuat collection baru. Sebagai gantinya, inline lebih banyak menggunakan resource karena dipaksakan untuk mencetak hasil yang sekian banyak dalam waktu singkat.

3.2 Fungsi MR2

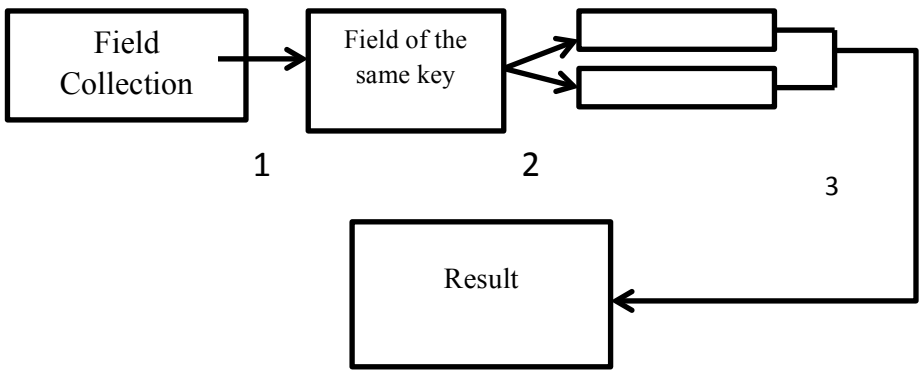
Fungsi MR2 menerapkan Javascript kedalam fungsi MapReduce. Disini, fungsi Map dan fungsi Reduce akan di-*define* terlebih dahulu sebelum dipanggil oleh fungsi MapReduce.

```
map2 = function(){ emit(this.name,
this.grades[0]) }

reduce2 = function(key, value){  return value[0];
}
db.building.mapReduce(map2, reduce2,  { out:
"Grading" })
```

Fungsi MR2 juga mengelompokkan berdasarkan *key* yang diberikan di fungsi *map,* yaitu field ‘nama’ yang diubah oleh fungsi reduce menjadi field ‘_id’ di *collection* yang baru.

Proses MapReduce diatas dapat diilustrasikan seperti ini:



Gambar 3-1: Ilustrasi MR2

Ilustrasi diatas berjalan seketika fungsi *result2* dieksekusi oleh MongoS. Berikut adalah penjelasan dari alur fungsi tersebut:

Tahap 1 adalah bentuk dokumen awal yang kemudian akan dilakukan agregasi berdasarkan key yang ditentukan di fungsi map2, yaitu field 'nama' dan 'grades'. Kemudian pada tahap 2, fungsi map2 akan mengeksekusi fungsi *emit* yang akan mengelompokkan key berdasarkan value yang ditentukan di fungsi map2. Setelah itu, fungsi reduce2 akan mengelompokkan key hasil dari fungsi map tersebut terhadap masing-masing valuenya. Lalu pada tahap 3, output dari fungsi reduce2 akan disimpan di collection baru bernama "Grading". Jika collection bernama "Grading" sudah ada didalam database, maka hasil MR2 yang paling baru akan menggantikan isinya.

3.3 Fungsi MR3

Berikut adalah fungsi MapReduce ketiga yang akan diuji:

```
db.building.mapReduce(function() {
emit(this.restaurant_id, this.name) },

function(key,values) { return values[1] }, { out :
{ inline : true }});
```

Fungsi map akan melakukan mapping terhadap field restaurant_id ke field bernama name. Fungsi reduce akan mengembalikan nilai value yang sudah dipasangkan berdasarkan key, kemudian fungsi building.mapReduce akan meng-agregasikan nama restoran berdasarkan key restaurant_id dan seluruh hasilnya langsung diprint di mongo shell.

3.4 Fungsi MR4

Fungsi MapReduce ini akan melakukan map terhadap field *borough* (wilayah) sebagai key dan field *name* sebagai value. Fungsi reduce kemudian akan mengembalikan pasangan key/value yang kemudian akan dilakukan query untuk mencari restoran yang menyediakan masakan 'Bakery' yang diberikan *grade* A oleh pengunjung.

```
db.building.mapReduce(function() {          emit(this.cuisine, 1);   },

function(key, values) {

return Array.sum(values);    },

{ out : { inline : 1 }, query: {cuisine: 'Italian', 'grades.grade': 'A'}});
```

Nilai value yang akan dikembalikan oleh fungsi reduce harus diberikan index array karena fungsi mapReduce milik MongoDB belum mampu mengembalikan nilai value dalam bentuk array [1]. Dengan memberikan index pada value, maka value yang akan di keluarkan oleh fungsi reduce tidak akan berupa array, melainkan string, atau sesuai dengan tipe data value tersebut.

3.5 Fungsi MR5

Fungsi map pada MR5 melakukan proses mapping document berdasarkan key ‘nama’, ke value ‘street’. Street berada dibawah field address sebagai embedded document bersama dengan field building, zipcode, dan koordinat gedung, karenanya, field induk juga harus dipanggil untuk menjadikan field address sebuah value.

```
db.building.mapReduce(function() {

emit(this.name, this.address.street)
},      function(key,values) { return
values[1]},       { out : { inline : 1 }, query:
{cuisine: 'Italian', 'grades.grade': 'C'}});
```

Pada dokumen akan dilakukan agregasi pasangan key/value name dan street. Kemudian akan diquery untuk mendapatkan informasi mengenai restoran yang menyajikan masakan Itali dan diberi nilai C oleh pengunjung.

3.6 Fungsi MR 6

Fungsi MR 6 melakukan agregasi terhadap field *borough* dan field *name*. Pada MR6 terdapat query untuk melakukan pencarian terhadap nama restoran bernama “RAOS”, yang sebelumnya sudah dimasukkan ke collection building, di collection baru bernama “find” yang merupakan hasil dari fungsi mapReduce.

```
var map6 = function() {
 var key = this.restaurant_id;
var    boro = this.borough;
 var  name = this.name;
   var value = {    rest_id: this.restaurant_id,                          boro: this.borough,
addr: this.name,        };
emit( key, value ) }        ;

var reduce6 = function(key, value) {                          var reducedObjects = {
rest_id: key,
vals: value,                                  };
 return reducedObjects; };

db.building.mapReduce( map6, reduce6, {   query: { name: "RAOS" },    out: "Find"   } )
```

IV. PENGUJIAN

Pengujian dimulai dengan melakukan validasi data terlebih dahulu. Setelah mengetahui bahwa data JSON sudah valid, barulah mongoimport dilakukan. Mongoimport berfungsi untuk melakukan pengisian terhadap database dan collection yang dituju menggunakan dokumen dari file JSON.

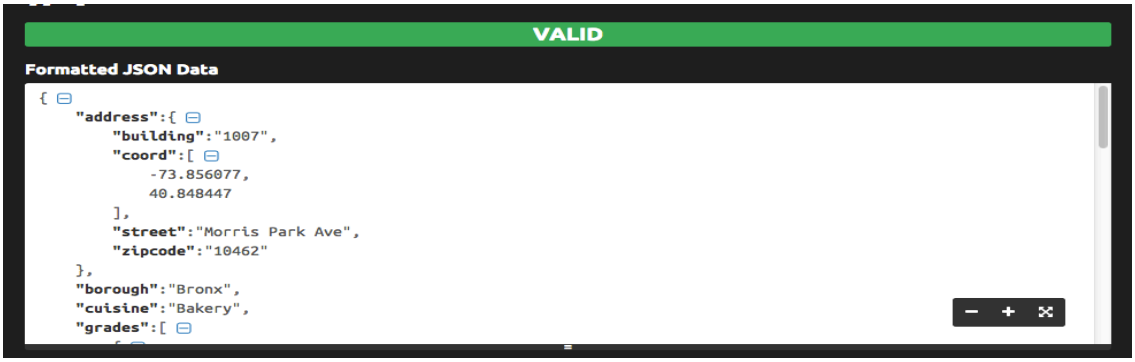
Pengujian dilakukan untuk dua kasus, yaitu pada lingkungan basisdata berbasis dokumen terdistribusi dan database lokal. Pengujian terhadap database lokal dilakukan dalam mesin yang sama dengan router pada lingkungan yang terimplementasi sharding.

Pengujian fungsi MapReduce dilakukan setelah kedua lingkungan yang berbeda itu selesai dibangun. Fungsi MapReduce akan dieksekusi 10 kali untuk mencari rata-rata response timenya. Setelah hasil dari Robomongo dicatat, dilakukan analisis terhadap *response time*-nya.

A. VALIDASI DATA

Validasi data dilakukan dengan menggunakan robomongo atau program yg disediakan oleh situs http://jsonformatter.curiousconcept.com/. Situs ini menerapkan format *JSON schema* untuk melakukan validasi

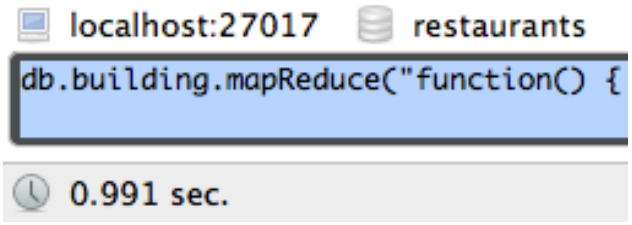
dokumen JSON dengan cara mendefinisikan struktur dari dokumen yang digunakan. Berikut adalah hasil validasi dari salah satu dokumen yang ada di collection building:



Gambar 0-1: Hasil Validasi Data

B. PENGUJIAN FUNGSI MAPREDUCE

Pengujian serta analisis performansi terhadap fungsi MapReduce dilakukan dengan membandingkan parameter *response time* MapReduce yang diimplementasikan ke sebuah *instance* MongoDB terhadap fungsi MapReduce yang sama, tetapi diimplementasikan ke sistem basisdata terdistribusi. *Response time* dapat diperoleh dari Robomongo setelah fungsi di *run*.



Gambar 4-2: Response time pada Robomongo

Fungsi MapReduce akan diuji sebanyak 10 kali agar waktu response time yang didapat lebih stabil, lalu dicari nilai rata-ratanya.

C. Spesifikasi Perangkat Keras dan Perangkat Lunak

Pengujian dilakukan dengan menggunakan computer dengan prosesor Intel Core i3 540 (3.07 GHz) dan 4 MB of RAM menggunakan sistem operasi Windows 7 Ultimate (64-bit) yang sudah terinstall MongoDB sebanyak 7 unit. OS harus merupakan OS 64-bit karena OS 32-bit tidak bisa melakukan sharding.

D. Dataset

Dataset yang digunakan adalah data restoran sebanyak 25.359 baris yang dapat diperoleh di https://docs.mongodb.org/

V. HASIL DAN ANALISIS

Pada sub-bab akan dibahas mengenai hasil pengujian yang dilakukan pada dua jenis implementasi MongoDB, localhost (satu *instance*) dan *sharded environment* (banyak instance/*distributed*).

A. Analisis dan Hasil Pengujian MR1

Berikut adalah beberapa hasil dari fungsi MR1:

```
{    "results" : [
 {          "_id" : "",    "value" : "Bronx"       },
{          "_id" : "07005",  "value" : "Missing" },
 {          "_id" : "10000", "value" : "Manhattan"},
```

Fungsi MR1 berhasil melakukan agregasi terhadap dokumen menggunakan field zipcode dan borough. Karena fungsi MR1 mengembalikan nilai hasil menggunakan *inline*, maka mongo shell langsung menampilkannya setelah eksekusi query selesai.

Pada hasil MR1, field '_id', yang berfungsi sebagai key dari fungsi MR1, diberikan value dari field zipcode milik document pada collection building, sedangkan field 'value' diisi dengan nilai dari field borough.
Berikut merupakan hasil perhitungan *response time* (dalam satuan detik) dari MR1 yang dilakukan oleh Robomongo.

Tabel 4-2: Hasil perhitungan fungsi MR1

Eksekusi ke-	**Tanpa Sharding**	**Distributed**		
		2 shard	3 shard	5 shard
1	0.773	0.236	0.155	0.177
2	0.38	0.204	0.172	0.163
3	0.382	0.194	0.18	0.118
4	0.383	0.223	0.153	0.133
5	0.37	0.236	0.168	0.128
6	0.367	0.229	0.176	0.142
7	0.384	0.224	0.215	0.144
8	0.368	0.258	0.179	0.124
9	0.426	0.195	0.178	0.171
10	0.384	0.218	0.229	0.157
Rata-rata	0.421	0.2217	0.1805	0.1457

Dapat dilihat bahwa Query MR1 bekerja lebih cepat pada sistem terdistribusi. Hal ini berkaitan dengan penjelasan di bagian landasan teori mengenai *sharding* yang menyatakan bahwa pada sharding, request atau query akan dilakukan oleh setiap shard yang ada, dimana shard tersebut tersimpan potongan collection yang dibagi secara merata. Artinya, query mapReduce pada MongoDB biasa memberikan hasil lebih lambat karena harus melakukan pengecekan secara menyeluruh terhadap collection yang masih utuh. Hal yang sama dapat dikatakan pada implementasi MapReduce di lingkungan terdistribusi yang hanya menggunakan 1 shard saja.

B. Analisis dan Hasil Pengujian MR2

Hasil dari fungsi MR2 akan tersimpan di collection bernama Grading yang baru dibuat oleh MongoDB setelah fungsi MR2 selesai dieksekusi. Berikut adalah beberapa dokumen dari collection Grading:

```
{ "_id" : "#1 Garden Chinese", "value" : { "date" :
ISODate("2014-10-29T00:00:00Z"), "grade" : "A",
"score" : 10 } }
{ "_id" : "#1 Me. Nick'S", "value" : { "date" :
ISODate("2014-10-09T00:00:00Z"), "grade" : "A",
"score" : 10 } }
```

Berdasarkan hasil tersebut, fungsi MapReduce berhasil mengagregasikan *collection* 'building' berdasarkan value-nya, field *grades*.

Berikut adalah hasil dari perhitungan *response time* pada fungsi MR2 yang didapatkan dengan menggunakan Robomongo

Tabel 4-3: Hasil Perhitungan Query MR2 (detik)

Eksekusi ke-	**Tanpa Sharding**	**Distributed**		
		2 shard	3 Shard	5 Shard
1	1,769	1.179	0.778	0.692
2	1,111	0.785	0.687	0.667
3	1,091	0.738	0.743	0.669
4	1,11	0.79	0.637	0.679
5	1,065	0.815	0.645	0.693
6	1,054	0.857	0.758	0.711
7	1,085	0.779	0.806	0.707
8	1,057	0.894	0.634	0.697
9	1,085	0.827	0.655	0.73
10	1,092	0.769	0.945	0.685
Rata-rata	1.152	0.8433	0.7288	0.693

Single server memiliki response time yang lebih lama dibandingkan dengan pengeksekusian fungsi MR2 di lingkungan basisdata berbasis dokumen terdistribusi. Hasil ini sesuai dengan teori bahwa query pada *sharding* akan lebih cepat karena dieksekusi oleh tiap *shard*.

C. Analisis dan Hasil Pengujian MR3

Berikut adalah beberapa hasil dari fungsi MR3:

```
{    "results" : [
{  "_id" : "30075445",
 "value" : "Morris Park Bake Shop"       },
{  "_id" : "30112340",
"value" : "Wendy'S"       },
{  "_id" : "30191841",
"value" : "Dj Reynolds Pub And Restaurant"
```

Fungsi MR3 berhasil melakukan mapping terhadap nama restoran berdasarkan key restaurant_id yang dimana value lainnya sudah dihapus oleh fungsi reduce.

Berikut merupakan hasil dari perhitungan *response time* dari fungsi MR3 yang didapatkan dengan menggunakan Robomongo:

Tabel 0-4: Hasil Pengujian Response Time (detik) pada Query MR3

Eksekusi ke-	Tanpa Sharding	Distributed		
		2 Shard	3 Shard	5 shard
1	5.133	0.825	0.845	0.76
2	5.03	0.912	0.797	0.837
3	5.11	0.838	0.775	0.774
4	5.04	0.787	0.808	0.737
5	5.069	0.796	0.777	0.803
6	5.018	0.851	1.025	0.735
7	5.075	0.808	0.787	0.791
8	5.095	0.866	0.743	0.772
9	5.163	1.201	0.743	0.794
10	5.037	0.988	0.78	0.778
Rata-rata	5.077	0.8872	0.808	0.7781

Response time MR3 menunjukkan bahwa query MR3 memakan waktu yang jauh lebih lama disbandingkan dua query MapReduce sebelumnya. Hal ini membuktikan bahwa performa fungsi mapReduce sangat terpengaruh oleh panjangnya data yang dihasilkan. MR3 tidak memiliki subquery tambahan yang melakukan filtrasi terhadap dokumen yang akan diproses menggunakan fungsi Map dan fungsi Reduce. Hal ini mengakibatkan jumlah dokumen yang harus diproses oleh mongo shell sangat banyak dan membebani resource.

D. Analisis dan Hasil Pengujian MR4

Berikut adalah beberapa hasil dari fungsi MR4:


```
{       "results" : [
{"_id" : "Italian",
"value" : 2100
}       ],
"timeMillis" : 0204,
        "counts" : {
                "input" : 2100,
                "emit" : 2100,
                "reduce" : 21,
                "output" : 1
        },      "ok" : 1,   }
```

Hasil tersebut menyatakan bahwa terdapat 2100 restoran Itali di collection building yang diberii nilai A. TimeMillis merupakan *response time,* sedangkan *counts* menunjukkan jumlah dokumen yang diproses oleh mapReduce. Input adalah jumlah yang diterima fungsi map. Jumlah tersebut sebelumnya sudah tersaring oleh subquery pada MR2.

Berikut merupakan hasil dari perhitungan *response time* dari Query MR4 yang didapatkan dengan menggunakan aplikasi Robomongo.

Tabel 0-5: Hasil Perhitungan Response Time (detik) Query MR4

Pengujian Ke-	Tanpa Sharding	Distributed		
		2 Shard	3 Shard	5 shard
1	0,204	0.075	0.058	0.04
2	0,049	0.051	0.055	0.056
3	0,054	0.069	0.049	0.059
4	0,027	0.074	0.05	0.058
5	0,049	0.052	0.063	0.055
6	0,046	0.063	0.066	0.048
7	0,039	0.059	0.061	0.065
8	0,05	0.057	0.051	0.067
9	0,042	0.047	0.046	0.049
10	0,037	0.071	0.048	0.045
Rata-rata	0,060	0.0618	0.0547	0.0542

Hasil dari perhitungan response time oleh Robomongo masih membuktikan bahwa performansi MapReduce lebih baik daripada implementasi pada local MongoDB. Meskipun demikian, pada fungsi hasil pengujian MR4 ini juga dapat dilihat bahwa perhitungan *average response time* pada database local tidak berbeda jauh dari implementasi pada sistem basisdata terdistribusi homogen. Hal ini didisebabkan oleh keluaran dari fungsi mapReduce yang sedikit karena sudah melalui proses filtering di bagian query fungsi MR4.

E. Analisis dan Hasil Pengujian MR5

Berikut adalah hasil dari fungsi MR5:

```
{    "results" : [
 { "_id" : "Abottega",          "value" : "Bedford
Street"       },
 { "_id" : "Acappella Restaurant",
"value" : "Hudson Street"       },  …
```

Fungsi MR5 berhasil mendapatkan nama dan juga alamat dari restoran. Field _id merupakan key, yaitu nama restoran sedangkan field value adalah didapat dari field address.street pada collection 'building'.

Hasil dari perhitungan *response time* dari fungsi MR5 yang didapatkan dengan menggunakan Robomongo adalah sebagai berikut:

Tabel 4-6: Hasil Perhitungan Response Time (detik) MR5

Pengujian Ke-	Tanpa Sharding	Distributed		
		2 shard	3 shard	5 shard
1	0,057	0.055	0.069	0.075
2	0,055	0.059	0.068	0.062
3	0,141	0.056	0.055	0.068
4	0,069	0.054	0.047	0.047
5	0,055	0.061	0.044	0.043
6	0,054	0.066	0.065	0.046
7	0,053	0.055	0.051	0.069
8	0,055	0.062	0.058	0.049
9	0,055	0.047	0.052	0.05
10	0,052	0.064	0.055	0.049
Rata-Rata	0,065	0.0579	0.0564	0.0558

Dari tabel diatas, rata-rata *response time* mapReduce yang diimplementasi pada database local lebih lama dibandingkan rata-rata *response time* di distributed database menggunakan *2 shard*.

F. Analisis dan Hasil Pengujian MR6

Berikut adalah hasil dari fungsi MR6:

```
{    "_id" : "1.0000000",
"value" : {       "rest_id" : "31231231",
"boro" : "Buah Batu",
 "name" : "RAOS "    } }
```

Hasil tersebut mengindikasikan bahwa fungsi berhasil mendapatkan hasil yang diinginkan, yaitu informasi mengenai lokasi dan id restoran "RAOS Restaurant". Adapun hasil dari perhitungan *response time* dari fungsi MR6 yang didapatkan oleh Robomongo adalah sebagai berikut:

Tabel 4-7: Hasil Perhitungan Response Time Query MR6 (detik)

Pengujian Ke-	Tanpa Sharding	Distributed		
		2 shard	3 shard	5 shard
1	0,156	0.071	0.052	0.05
2	0,069	0.064	0.047	0.058
3	0,072	0.061	0.04	0.047
4	0,064	0.061	0.052	0.067
5	0,066	0.055	0.063	0.054
6	0,086	0.06	0.052	0.056
7	0,071	0.062	0.063	0.044
8	0,061	0.035	0.084	0.06
9	0,106	0.061	0.05	0.064
10	0,062	0.06	0.055	0.057
Rata-Rata	0,081	0.059	0.0558	0.0557

Hasil tersebut menjelaskan bahwa mapReduce dapat memberikan hasil yang lebih cepat di lingkungan basisdata terdistribusi.

G. Throughput

Mayoritas dari nilai rata-rata hasil pengujian Query MR1 hingga MR6 menunjukkan bahwa MapReduce MongoDB lebih cepat memberikan hasil pada sistem terdistribusi dibandingkan dengan implementasinya di single server. Lingkungan terdistribusi dapat meningkatkan throughput hingga sebesar 62%. Hal ini dikarenakan oleh sifat distribusi dari sharding yang memungkinkan pengerjaan fungsi map dan reduce untuk dikerjakan di masing-masing shard, bukan dikerjakan di satu *site* saja.

Keunggulan sharding dengan menggunakan lebih banyak shard sangat terlihat di query MR2. Disini, rata-rate *response time* pada sharding dengan 1 shard bahkan lebih lambat dari MongoDB tanpa sharding.

Performansi MapReduce juga bergantung kepada penggunaan resource. Dari hasil MR3, kita dapat melihat bahwa *response time* yang sangat lambat jika dibandingkan dengan fungsi lainnya. Hal ini terjadi karena MR3 tidak menggunakan subquery untuk mengeleminasi data yang tidak dibutuhkan. Dari response time, kita dapat mencari nilai *throughput* dengan cara menggunakan rumus:

$$Throughput\ (Request/sec) = \frac{1}{Response\ Time} \qquad (4.1)$$

Response time yang digunakan adalah *average response time.* Dengan menggunakan rumus tersebut, didapatkan hasil sebagai berikut:

Tabel 4-8: Throughput masing-masing fungsi (request/detik)

Nama Fungsi	Tanpa Sharding	Distributed		
		2 shard	3 shard	5 shard
MR1	2.375	4.510	5.540	6.863
MR2	0.868	1.186	1.372	1.443
MR3	0.196	1.127	1.237	1.285
MR4	16.66	16.87	18.281	18.45
MR5	15.38	17.271	17.730	17.921
MR6	12.34	16.949	17.921	17.953

Pada grafik di gambar 4-10 juga dapat dilihat bahwa semakin banyak shard, maka nilai throughput akan meningkat. Hal ini terjadi karena semakin banyak shard, maka semakin sedikit pula dokumen yang harus diproses oleh sebuah shard.

Dari tabel 4-8 dapat dilihat bahwa nilai throughput pada query MR3 di lingkungan terdistribusi tidak lebih baik dibandingkan dengan query MR5 di single server. Hal ini disebabkan oleh penggunaan subquery di MR5. Subquery mempercepat proses query MapReduce karena dokumen-dokumen pada *collection 'building'* sudah diseleksi terlebih dahulu sebelum diterima oleh fungsi map. Sedangkan pada MR3 yang terdistribusi, seluruh shard diharuskan melakukan pemrosesan query karena collectionnya tidak diseleksi oleh subquery, sehingga nilai *throughput*-nya jauh lebih sedikit dibandingkan dengan MR5 di *single server*.

VI. KESIMPULAN DAN SARAN

A. KESIMPULAN

Berdasarkan analisis dan pengujian, didapatkan kesimpulan sebagai berikut:

1. Lingkungan basisdata berbasis dokumen yang terdistribusi secara homogen yang dibangun menggunakan *sharding* meningkatkan performansi MapReduce.
2. Semakin banyak shard yang digunakan dalam suatu lingkungan terdistribusi semakin meningkat performa query MapReduce.
3. MapReduce lebih efektif ketika digunakan lebih banyak subquery untuk mengeliminasi key dan value yang tidak dibutuhkan.

B. SARAN

Berikut adalah beberapa saran dari penulis jika ingin melakukan penelitian mengenai MongoDB:

1. Pada MongoDB, terdapat fungsi selain MapReduce yang fungsinya hampir sama, yaitu *aggregation framework.* Akan menarik untuk menelitinya lebih dalam dan mencari tahu performansinya di lingkungan yang terimplementasi *sharding*.
2. Mengimplementasikan Replica Set pada Sharding untuk meneliti *availability* yang diberikan oleh metode Replica Set.
3. Melakukan penelitian terhadap MapReduce terhadap data real-time. Framework MapReduce yang diciptakan oleh Google dibuat dengan tujuan untuk mengatasi masalah data real-time yang sangat besar. Akan sangat menarik jika dapat melakukan penelitian terhadapnya.

REFERENSI

[1] MongoDB, Inc. (2014, November) MongoDB. [Online]. http://docs.mongodb.org/

[2] Google Inc., "Introduction to Parallel Programming and MapReduce," 2007.

[3] Eva Fransisca S, *Implementasi dan Analisis Performansi Heterogeneous Distributed Database pada Database as a Service*., 2013.

[4] J Lin and Dyer C, "Data-Intensive Text Processing with MapReduce," , 2010.

[5] Michael Meadway, *STORING AND RETRIEVING OBJECTS ON A COMPUTER NETWORK IN A DISTRIBUTED DATABASE*., 2014.

[6] Howard Karloff and Siddarth Suri, "A Model of Computation for MapReduce," 2014.

[7] Muhammad Shahrizal Prabowo, *Migrasi Data Dari Basisdata Relasional ke Basisdata Dokumen dan Framework MapReduce*., 2012.

[8] Peter Buneman, "Semistructured Data," 1997.

[9] M. Tamer Öszu and Patrick Valduriez, *Principles of Distributed Database Systems*, 3rd ed. New York: Springer, 2011.

[10] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *OSDI 2004*, 2004.