Pencarian Data Tersebar Menggunakan Algoritma *Divide and Conquer* dengan Model Pemrosesan MapReduce

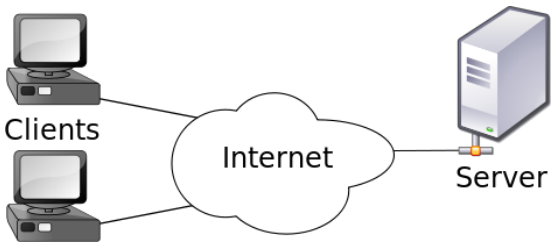
Setyo Legowo (13511071)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*setyo.legowo@students.itb.ac.id*

***Abstrak*—Kian tahun kebutuhan informasi kian meningkat. Meningkatnya proses berbagi informasi diantara mesin berimplikasi pada kebutuhan kemampuan mesin yang dapat melayani kepada mesin yang ingin berkomunikasi. Namun, peningkatan kemampuan mesin sudah tidak relevan lagi karena terbatasnya kemampuan perangkat keras yang ada, terutama pada penyimpanan informasi sebanyak-banyaknya. Oleh karena itu diperlukan sebuah model pemrosesan baru yang dapat mengolah informasi dari berbagai mesin agar mendapatkan informasi yang lebih baik. Salah satu tren yang sekarang ini digunakan oleh perusahaan besar IT adalah model pemrosesan MapReduce. Strategi yang digunakan oleh MapReduce mengimplementasi strategi divide and conquer. MapReduce membagi persoalan menjadi beberapa upa-persoalan lalu diolah oleh mesin-mesin di suatu kawasan lalu mengumpulkannya lagi dan menghasilkan keluaran yang optimal.**

Kata kunci*—MapReduce, *divide and conquer*, distribusi, *query

I. PENDAHULUAN

Di dalam jaringan komputer, setiap komputer dapat terhubung untuk dapat saling berkomunikasi. Salah satu manfaat yang bisa didapatkan dari komputer yang saling terhubung adalah setiap pengguna dapat berbagi informasi. Namun untuk menunjang informasi yang didapat dengan baik dan sesuai dengan permintaan pengguna maka harus terdapat komputer yang bisa melayani dengan baik. Beberapa faktor yang mempengaruhinya diantaranya kecepatan akses dalam mendapatkan informasi, informasi yang didapat adalah tepat atau sesuai, dan biaya yang rendah.



Gambar 1.1 Arsitektur *Client-Server*

Di era zaman globalisasi ini, berbagi informasi tidak hanya antara satu komputer dengan komputer yang lain saja, tapi satu komputer yang dapat melayani lebih dari satu komputer seperti yang terlihat pada Gambar 1.1. Komputer tersebut harus memiliki kemampuan khusus dalam melayani permintaan informasi dari komputer yang lain. Kita sebut komputer ini adalah sebuah komputer *server*. Semakin banyak komputer yang mengakses ke *server* maka kemampuan *server* juga harus semakin meningkat. Supaya *server* tidak terlalu terbebani oleh koneksi komputer yang bebannya semakin meningkat setiap waktunya, maka solusi yang ada adalah membagi-bagi persoalan ke lebih dari satu *server*. Bentuk persoalan pada jaringan dapat berbentuk berbagai macam persoalan. Pada mesin pencari, persoalan yang digunakan adalah informasi yang cocok sesuai frase yang diinginkan. Pada perangkat lunak proyek komputasi distribusi memiliki berbagai macam persoalan yang dapat diselesaikan tergantung jenis proyek yang digunakan. Contoh jenis proyek komputasi terdistribusi seperti analisis prediksi model iklim, memonitor objek asteroid terdekat dengan bumi yang berbahaya, kalkulasi protein, dan masih banyak lagi.

Untuk dapat membagi-bagi persoalan ke lebih dari satu *server* dengan baik maka diperlukan beberapa strategi algoritma yang dapat menghasilkan informasi yang diinginkan oleh pengguna. Strategi algoritma yang akan digunakan di dalam makalah ini untuk menyelesaikan masalah di atas adalah *divide and conquer*.

Untuk dapat menggambarkan strategi yang digunakan maka di dalam makalah ini menggunakan salah satu model pemrosesan data dalam pemrosesan data terdistribusi yaitu MapReduce. MapReduce saat ini digunakan untuk dapat memproses basis data yang ukurannya sudah mencapai Petabytes daripada RDBMS (Rational Database Management System) yang mampu memproses dalam Gigabytes.

Divide and conquer berguna untuk membagi persoalan menjadi beberapa persoalan sehingga cukup kecil untuk dilakukan komputasi oleh *server*. Lalu solusi dari persoalan tersebut dari masing-masing *server* digabung sehingga mendapatkan solusi dari persoalan semula. Algoritma yang digunakan untuk menyelesaikan persoalan ini pada *divide and conquer* adalah *string matching*. *String matching* digunakan untuk menemukan frase yang cocok pada pemrosesan di basis data. *String matching* yang digunakan pada MapReduce tidak akan dibahas secara mendalam di dalam makalah ini.

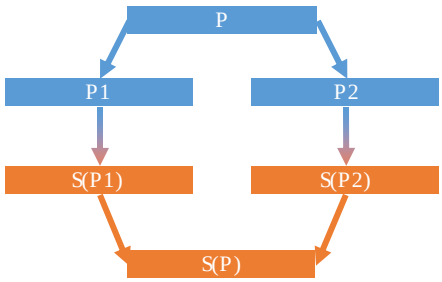
II. DASAR TEORI

2.1 Divide and Conquer

Divide and conquer adalah metode pemecahan masalah yang bekerja dengan membagi masalah (*problem*) menjadi beberapa upa-masalah (*sub problem*) yang lebih kecil, kemudian menyelesaikan masing-masing upa-masalah secara independen, dan akhirnya menggabung solusi masing-masing upa-masalah sehingga menjadi solusi masalah semula. Lebih persisnya, algoritma *divide and conquer* disusun oleh tiga proses utama:

- *Divide*: membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran kecil (idealnya berukuran hampir sama),
- *Conquer*: memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif), dan
- *Combine*: menggabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula.

Obyek permasalahan yang dibagi adalah masukan (*input*) atau instan yang berukuran n. Masukan tersebut mungkin berupa tabel (larik), matriks, eksponen, dan sebagainya, bergantung pada masalahnya. Tiap-tiap upa-masalah mempunyai karakteristik yang sama (*the same type*) dengan karakteristik masalah asal, sehingga metode *divide and conquer* lebih natural diungkapkan dalam skema rekursif. Tetapi tidak semua fungsi rekursif adalah algoritma divide and conquer.



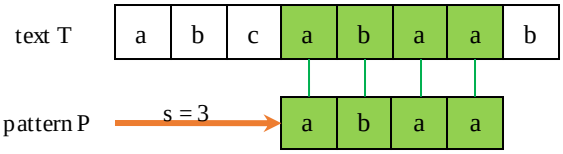
Gambar 2.1 Skema ilustrasi pembagian persoalan pada *divide and conquer*

Metode *divide and conquer* memberikan dua keuntungan. Pertama, metode ini menyediakan pendekatan yang sederhana untuk memecahkan masalah yang secara konseptual sulit. Kedua, jika solusi masalah sudah diketahui secara pasti (seperti masalah pengurutan, masalah pengangkatan bilangan bilangan secara *brute force*), *divide and conquer* dapat secara substansial mengurangi biaya (*cost*) komputasi (kompleksitas).

2.2 String Matchings

Program pengelola teks sering kali digunakan untuk mencari semua string yang mirip dengan sebuah pola tertentu (pattern) pada teks. Biasanya teks adalah dokumen yang sedang digunakan dan mencari apakah pattern tersebut berada pada dokumen yang sedang dibuka. Algoritma yang efisien untuk permasalahan ini yang kita sebut sebagai *string matching* dapat meningkatkan program pengelola teks menjadi lebih responsif.

Masalah *string maching* secara formal dapat dideskripsikan sebagai berikut. Asumsikan bahwa teks berada pada suatu larik T[1..n] dengan panjang teks adalah *n* dan pattern adalah sebuah larik P[1 .. *m*] dengan panjang $m \leq n$. Lalu kita asumsikan bahwa elemen dari P dan T adalah karakter yang terdaftar pada Σ. Contoh isi Σ = {0, 1} atau Σ = {a, b, ..., z}. Larik dari karakter pada P dan T sering disebut sebagai karakter teks.



Gambar 2.2 Sebuah contoh masalah *string matching*

Berdasarkan Gambar 2.2, dapat dikatakan P berada pada posisi s pada teks T (atau, pattern P dimulai dari posisi s + 1 dari teks T) jika $0 \leq s \leq n - m$ dan $T[s + 1 \ldots s + m] = P[1 \ldots m]$. Jika P berada pada shift s pada T, maka disebut sebagai shift yang valid; jika tidak maka s berada pada shift yang tidak valid. Masalah *string matching* merupakan masalah untuk mencari semua shift yang valid untuk pattern P yang berada pada teks T.

2.3 MapReduce

MapReduce merupakan paradigma dalam pemrosesan data yang diletakkan pada ratusan komputer. MapReduce menyediakan sebuah model pemrograman yang mengabstraksikan masalah pembacaan dari disk dan penulisan pada disk dengan mentransformasikan pembacaan dan penulisan menjadi himpunan dari kunci dan nilainya.

Dalam komputasi MapReduce membutuhkan sebuah himpunan kunci/nilai untuk masukannya dan menghasilkan sebuah himpunan kunci/nilai untuk keluarannya. Komputasi tersebut dapat dibagi menjadi dua fungsi: *Map* dan *Reduce*.

Map, didefinisikan dengan sebuah pasangan masukan kunci/nilai dan keluaran fungsi tersebut adalah himpunan dari pasangan-pasangan kunci/nilai (*intermediate key*). Pustaka MapReduce akan mengelompokkan semua pasangan-pasangan kunci/nilai yang diasosiasikan dengan kunci *intermediate I* yang sama dan memberikannya pada fungsi *Reduce*.

Fungsi *Reduce* didefinisikan dengan mengambil sebuah kunci *intermediate I* dengan himpunan nilai untuk kunci tersebut. Nilai tersebut digabung seluruhnya untuk membentuk nilai yang mungkin sebuah himpunan nilai yang lebih kecil. Biasanya hanya 0 atau 1 himpunan nilai yang keluar untuk setiap fungsi *Reduce* dipanggil. Nilai *intermediate* yang diberikan masuk ke dalam fungsi *reduce* melalui iterator. Iterator tersebut digunakan untuk dapat mengelola nilai yang terlalu besar untuk disimpan di dalam

memori dalam bentuk *list*.

Untuk dapat menggambarkan MapReduce diberikan sebuah contoh. Diberikan sebuah permasalahan tentang menghitung jumlah kata yang ada di dalam dokumen yang jumlah koleksinya sangat banyak. Lalu dibuat sebuah *pseudo-code* untuk menyelesaikan permasalahan tersebut pada Tabel 2.1.

Pada tabel tersebut fungsi `map` mengeluarkan setiap kata yang akan dihitung dan diasosiasikan pada jumlah dari suatu kejadian (hanya `'1'` pada contoh tersebut). Fungsi `reduce` menjumlahkan semua yang dikeluarkan oleh fungsi map untuk setiap katanya.

```
map(String key, String value):
   // key: document name
   // value: document contents
   for each word w in value:
      EmitIntermediate(w, "1");

reduce(String key, Iterator values):
   // key: a World
   // values: a list of counts
   int result = 0;
   for each v in values:
      result += ParseInt(v);
   Emit(AsString(result));
```

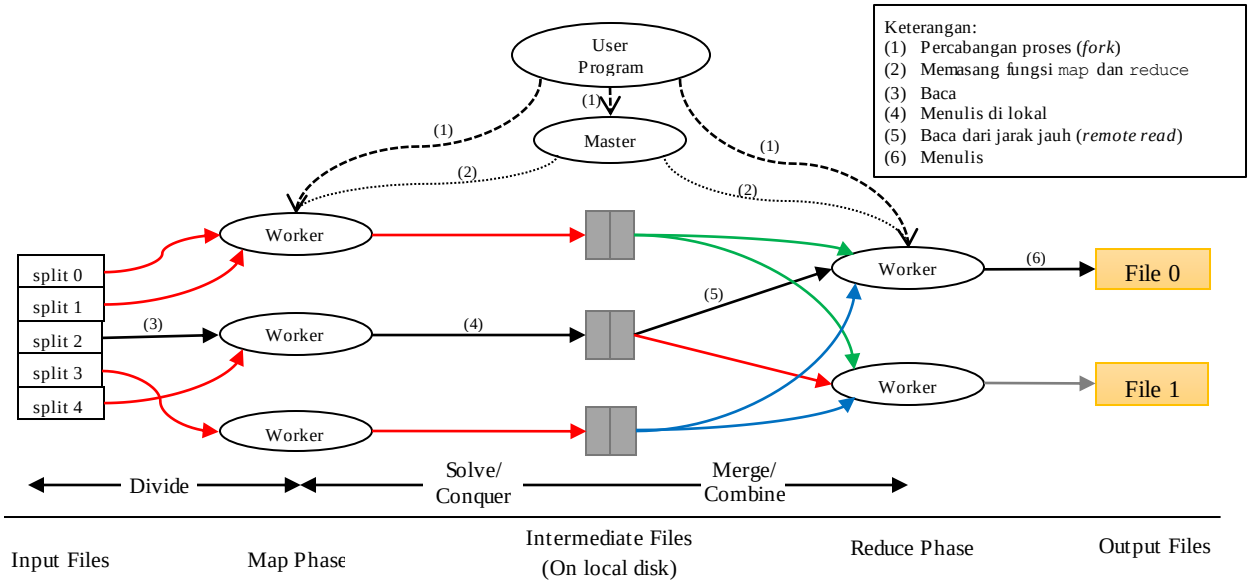
Tabel 2.1 *Pseudo-code* fungsi `Map` dan `Reduce`

Dari pseudo-code yang terdapat pada Tabel 2.1, secara konsep fungsi *map* dan *reduce* yang ditulis mempunyai tipe seperti berikut:

Map `(k1,v1)` → `list(k2,v2)`

Reduce `(k2,list(v2))` → `list(v2)`

Dengan kata lain, kunci masukan dan nilainya dibuat dari domain yang berbeda dibandingkan dengan keluarannya dalam bentuk kunci dan nilanya. Selain itu, kunci *intermediate* dan nilainya berada pada domain yang sama dengan kunci keluarannya dan nilainya.



Gambar 3.1 Ilustrasi eksekusi MapReduce dan bagian yang termasuk *divide and conquer*

III. Cara Kerja

3.1 Ilustrasi Eksekusi MapReduce

Fungsi Map dipanggil secara terdistribusi ke mesin yang banyak yang secara otomatis membagi data masukan ke dalam *M* bagian. Masing-masing bagian dapat diproses oleh mesin-mesin yang berbeda secara paralel. Fungsi *reduce* dipanggil secara terdistribusi dengan membagi-bagi kunci *intermediate* ke dalam *R* bagian menggunakan sebuah fungsi partisi (contoh *hash(key)* **mod** *R*). Banyaknya partisi (*R*) dan fungsi partisi dibentuk tersendiri (tidak disediakan oleh sistem).

Gambar 3.1 memperlihatkan salah satu implementasi dari keseluruhan cara kerja dalam mengoperasikan MapReduce. Ketika program yang dibuat memanggil fungsi MapReduce, di bawah ini diperlihatkan urutan dari aksi yang terjadi (label yang dinomori pada Gambar 3.1 berhubungan dengan urutan aksi di bawah ini):

1. Pustaka MapReduce di dalam program pertama membagi masukan (berupa *files*) menjadi M bagian. Biasanya ukuran setiap bagian sebesar 16 megabytes hingga 64 megabytes (MB) (dapat diatur oleh pemrogram melalui parameter opsional). Lalu MapReduce menjalankan salinan programnya (*fork*) ke mesin-mesin yang berada pada sebuah *cluster*.
2. Setiap salinan dari program unik – sebut saja program master. *Worker* yang sedang tidak bekerja diberikan sebuah pekerjaan oleh program master, yakni memberikan *M map* tugas dan *R reduce* tugas yang diberikan ke *Worker*.
3. Sebuah *Worker* yang diberikan tugas untuk melakukan tugas map membaca isi dari masukan yang masuk ke dalam *Worker*. *Worker* tersebut mengurai pasangan kunci/nilai yang akan dikeluarkan dari data masukan dan menghasilkan

setiap pasang yang didefinisikan oleh program di fungsi map. Pasangan kunci/nilai *intermediate* dihasilkan oleh fungsi map disimpan pada *buffer* di dalam memori mesin.

4. Secara periodik, pasangan kunci/nilai di dalam *buffer* ditulis ke *disk* lokal, lalu dilakukan partisi menjadi R bagian oleh fungsi partisi. Lokasi dari pasangan *buffer* yang berada pada *disk* lokal diberikan kepada program master yang bertanggung jawab untuk meneruskan lokasi tersebut ke *Worker* yang diberikan tugas melakukan fungsi *reduce*.
5. Ketika *Worker* yang melakukan fungsi *reduce* diberi tahu oleh program master tentang lokasi tersebut, *Worker* melakukan RPC (*Remote Procedure Call*) untuk membaca data *buffer* dari *disk* lokal yang dimiliki oleh *Worker* fungsi map. Ketika suatu *Worker* fungsi *reduce* selesai membaca semua data *intermediate*, dilakukan proses pengurutan berdasarkan kunci *intermediate* untuk semua data dan mengelompokkan kunci yang sama secara bersamaan. Proses pengurutan di tahap ini penting karena terdapat banyak kunci fungsi map yang masuk ke dalam tugas *reduce* yang sama. Jika data yang *intermediate* sudah terlalu besar untuk ditampung ke dalam memori, maka akan dilakukan pengurutan eksternal.
6. *Worker* fungsi *reduce* melakukan terasi ke semua data yang *intermediate* dan untuk setiap kunci *intermediate* unik yang ditemukan, Worker fungsi *reduce* memasangkan kunci dengan himpunan nilai untuk kunci yang bersangkutan disesuaikan dengan fungsi *reduce* dari program. Hasil yang dikeluarkan oleh fungsi *reduce* ditambahkan ke hasil akhir dari fungsi *reduce* tersebut yang dipartisi.
7. Ketika semua tugas fungsi map dan *reduce* telah selesai, program master memanggil program yang memanggilnya. Pada titik ini, pemanggilan MapReduce di dalam program yang dibuat berakhir dan kembali ke program yang dibuat.

Setelah selesai dan berhasil menggunakan MapReduce, hasil dari MapReduce yang dieksekusi berada pada R (satu untuk setiap tugas fungsi *reduce*, dengan nama yang didefinisikan di dalam kode program yang dibuat). Biasanya, hasil yang dikeluarkan oleh MapReduce tidak dimampatkan lagi menjadi satu – sering kali hasilnya sebagai masukan untuk pemanggilan MapReduce selanjutnya, atau digunakan ke program aplikasi terdistribusi lainnya yang mempunyai hak atas hasil yang dihasilkan oleh fungsi MapReduce yang dipanggil yakni membagi-bagi hasil menjadi beberapa bagian lagi.

3.2 Ilustrasi dengan Divide and Conquer

Dari fungsi pada MapReduce yang di ilustrasikan pada Gambar 3.1, dapat diketahui bahwa fungsi *divide* berada pada fungsi map, fungsi *conquer* berada pada perantara fungsi map dengan *reduce*, dan fungsi *combine* berada pada fungsi *reduce*.

Fungsi *divide* pada MapReduce hanya dilakukan sekali untuk satu tugas yakni dengan membagi-bagi persoalan menjadi beberapa bagian yang ukurannya sama atau maksimum dengan ukuran tertentu. Ukuran terkecil untuk membagi persoalan pada MapReduce biasanya berukuran 16 megabytes hingga 64 megabytes (MB). Hal ini dilakukan untuk mendapatkan performasi yang baik. Ukuran terkecil dapat disesuaikan dengan ukuran satu blok pada disk sehingga mengurangi waktu untuk *seeking* pada disk.

Fungsi *conquer* atau *solving* dilakukan pada eksekusi satu bagian persoalan. Definisi fungsi untuk *conquer* atau *solving* pada MapReduce berada pada fungsi Map. Pada fungsi map diperlihatkan bagaimana satu bagian persoalan akan di selesaikan lalu dilakukan *emit* untuk dieksekusi tahap selanjutnya.

Fungsi *combine* pada MapReduce berada pada fungsi *reduce*. Fungsi *reduce* menggabungkan data *intermediate* yang diakses menggunakan RPC (dilakukan karena harus mengakses data dari mesin lain) lalu mengurutkannya berdasarkan kunci *intermediate* yang dibentuk pada fungsi map. Lalu, setiap kunci hanya dapat memiliki satu nilai sehingga fungsi *reduce* harus didefinisikan oleh pembuat program bagaimana data *intermediate* harus digabungkan.

IV. IMPLEMENTASI

Untuk dapat menggambarkan langkah kerja yang dilakukan oleh MapReduce, akan ditulis beberapa penggunaan fungsi *map* dan *reduce* untuk beberapa persoalan. Untuk dapat mensimulasikan kerja MapReduce, penulis menggunakan sebuah aplikasi manajemen basis data non-relasional yang dapat mengimplementasikan fungsi MapReduce yaitu CouchDB.

Sebagai data awal, terdapat sebuah mesin yang berisi tentang data nilai siswa dari suatu sekolah. Mesin tersebut berada pada sebuah *cluster* yang jumlah mesin pada *cluster* tersebut sebanyak satu.

Di dalam mesin tersebut berisi sebuah basis data nilai siswa dengan isi dari sebagian kolom yang ada sebagai berikut:

_id	Kode MP	Nilai	Keterangan
111210001	04001	78.00	T LULUS
111210002	04001	82.00	LULUS
111210003	04001	80.00	LULUS
111210004	04001	82.00	LULUS
111210005	04001	84.00	LULUS
111210006	04001	56.00	T LULUS
111210007	04001	74.00	T LULUS
111210008	04001	54.00	T LULUS

Tabel 4.1 Tabel data nilai siswa suatu sekolah

4.1 Mengeluarkan semua data

Untuk mengeluarkan semua data yang ada pada basis data, maka dibuat fungsi *map* dan *reduce* seperti berikut:

Fungsi map:

```
function(doc) {
      emit(null, doc)
}
```

Fungsi tersebut melakukan satu pembentukan kunci *intermediate* yaitu *null* (tidak didefinisikan) sehingga pada fungsi *reduce* tidak perlu melakukan pengurutan sehingga semua nilai yang ada pada setiap dokumen keluar seluruhnya. Dalam hal ini, tidak perlu mendefinisikan fungsi *reduce* karena tidak perlu digunakan.

Potongan hasil dari eksekusi pemanggilan data dapat dilihat pada tabel di bawah ini:


```
{"total_rows":8,"offset":0,"rows":[
   {"id":"111210001","key":null,"value":
     {"_id":"111210001",          "_rev":"1-
     c418415ccd666891b07d607bcc54ca7c",
     "KD_TAHUN_AJARAN":"19",
     "KD_TINGKAT_KELAS":"03",
     "KD_PROGRAM_PENGAJARAN":"3",
     "KD_ROMBEL":"2", ...
```


Tabel 4.2 Hasil eksekusi kasus 1

4.2 Menghitung jumlah baris data

Untuk menghitung jumlah dokumen yang ada pada basis data, maka dibuat fungsi *map* dan *reduce* sebagai berikut:

Fungsi map:

```
function(doc) {
      emit(doc._id, 1)
}
```

Pada fungsi tersebut memperlihatkan pembentukan kunci *intermediate* berdasarkan nomor yang unik untuk semua dokumen, dalam hal ini adalah Nomor Induk Siswa (NIS). Lalu nilai untuk setiap kuncinya diberi nilai 1 untuk memberi tahu sistem bahwa kunci tersebut mempunyai nilai 1 untuk setiap pembagian jumlah dokumen.

Fungsi Reduce:

```
function(key, values) {
      return sum(values)
}
```

Fungsi *reduce* tersebut memperlihatkan data masukan yang berasal dari data *intermediate* yang dijumlahkan nilainya. Karena nilai yang diberikan untuk setiap kunci adalah 1 maka jumlah dari kunci tersebut adalah 8.

Berikut adalah potongan hasil dari eksekusi pemanggilan data dapat dilihat pada tabel di bawah ini:


```
{"rows":[
    {"key":null,"value":8}
]}
```


Tabel 4.3 Hasil eksekusi kasus 2

4.3 Mengambil beberapa kolom pada dokumen

Untuk dapat mengambil beberapa kolom pada dokumen di basis data, maka dibuat fungsi *map* dan *reduce* sebagai berikut:

Fungsi map:

```
function(doc) {
      emit(null,
          {\"nis\":doc._id,
          \"kode_mata_pelajaran\":
          doc.KD_MATA_PELAJARAN,
          \"nilai\":doc.NILAI,
          \"Keterangan\":doc.KETERANGAN}
      )
}
```

Pada fungsi tersebut memberikan kunci untuk nilai dengan nilai *null* sehingga MapReduce tidak perlu perlu melakukan pengurutan pada data karena ingin mengeluarkan semua data dengan mengambil sebagian kolom di dokumen. Di fungsi tersebut diambil beberapa kolom yang ingin dihasilkan yaitu NIS yang berarti kunci dari dokumen tersebut, kode mata pelajaran untuk mata pelajaran tertentu, nilai untuk nilai dari mata pelajaran tersebut, dan keterangan untuk mata pelajaran tersebut.

Dalam masalah ini, tidak perlu mengimplementasikan fungsi *reduce*.

Potongan hasil dari eksekusi pemanggilan data dapat dilihat pada tabel di bawah ini:


```
{"total_rows":8,"offset":0,"rows":[
   {"id":"111210001","key":null,"value":{"nis
       ":"111210001","nilai":78,"Keterangan":"
       TIDAK LULUS"}},
   {"id":"111210002","key":null,"value":{"nis
       ":"111210002","nilai":82,"Keterangan":"
       LULUS"}},
   {"id":"111210003","key":null,"value":{"nis
       ":"111210003","nilai":80,"Keterangan":"
       LULUS"}},
...
```


V. KESIMPULAN

Untuk dapat menggali informasi yang dapat memproses persoalan dari informasi yang cukup besar diperlukan sebuah model yang mampu melakukan pemrosesan data yang tersebar di mesin-mesin yang berbeda. MapReduce merupakan sebuah model pemrosesan data tersebar yang mampu mengambil informasi dari mesin-mesin yang berbeda. Langkah kerja yang dilakukan MapReduce merupakan sebuah strategi algoritma yang dapat membagi masalah menjadi upa-masalah yaitu *divide and conquer*. Fungsi *divide* pada MapReduce disebut sebagai fungsi map. Fungsi *solve* dan *combine* pada MapReduce disebut sebagai fungsi *reduce*.

Untuk dapat membagi-bagi persoalan yang hadir dari salah satu mesin di dalam satu *cluster*, MapReduce membagi-bagi persoalan menjadi persoalan yang cukup kecil untuk dapat diproses oleh setiap mesin yang memiliki informasi di dalam mesin tersebut. Setiap bagian informasi menghasilkan sebagian hasil yang berupa data *intermediate*. Semua data tersebut yang berada pada *cluster* tersebut diambil oleh mesin tertentu. Mesin tersebut nantinya akan memproses data yang dilakukan oleh fungsi yang didefinisikan untuk memilih informasi yang ingin menjadi hasil akhir dari persoalan. Hasil dari MapReduce berupa kumpulan informasi yang disimpan di dalam sebuah *list*.

Di dalam makalah ini tidak dijelaskan secara lengkap penggunaan *string matching* dalam pencarian data di sistem basis data tersebar. Oleh karena itu, penulis mengharapkan di makalah selanjutnya dapat menjelaskan kegunaan *string matching* pada pencarian di sistem basis data tersebar.

VI. LAMPIRAN

Sistem manajemen basis data yang digunakan di dalam makalah ini yaitu CouchDB. CouchDB adalah sebuah basis data yang secara penuh mendukung untuk digunakan di dalam web. Data yang disimpan dibuat dalam bentuk dokumen JSON (JavaScript Object Notation). Untuk dapat mengakses dokumen dapat dilakukan melalui *browser* melalui protokol HTTP. *Query*, kombinasi, dan transformasi dokumen dilakukan dengan *javascript*. CouchDB bekerja dengan baik yang dapat digunakan oleh aplikasi *web* maupun Mobile. Dan juga data dapat disimpan secara terdistribusi atau di aplikasi secara efisien menggunakan CouchDB yang mudah berkembang sendiri dengan melakukan replikasi data.

Implementasi untuk pengujian MapReduce pada CouchDB pada makalah ini dilakukan dengan cara sebagai berikut:

1. Nama basis data yang digunakan adalah "data_test"
2. Terdapat 8 dokumen di dalam basis data.
3. Implementasi desain pada basis data didefinisikan sebagai berikut:

```
{
"_id": "_design/coba",
"_rev": "9-9307677500447c9a8fdec9f3f8537a9c",
"languange": "javascript",
"views": {
    "all": {
      "map": "function(doc){emit(null,doc)}"
    },
    "count": {
      "map": "function(doc){emit(doc._id,1)}",
      "reduce":  "function(keys,  values)  {
         return sum(values)}"
    },
    "several": {
      "map":"function(doc){emit(null,
         {\"nis\":doc._id,
         \"kode_mata_pelajaran\":
         doc.KD_MATA_PELAJARAN,
         \"nilai\":doc.NILAI,
         \"Keterangan\":doc.KETERANGAN})}"
    }
  }
}
```

4. Untuk mengakses setiap data uji yang dilakukan pada makalah ini dikunjungi melalui protokol HTTP.
   a. Pengaksesan data uji 1
      http://127.0.0.1:5984/nilai_siswa/_design/coba/_view/all
   b. Pengaksesan data uji 2
      http://127.0.0.1:5984/nilai_siswa/_design/coba/_view/count
   c. Pengaksesan data uji 3
      http://127.0.0.1:5984/nilai_siswa/_design/coba/_view/several

VII. UCAPAN TERIMA KASIH

Penulis ucapkan terima kasih kepada Tuhan Yang Maha Esa atas karunia-Nya yang diberikan telah membuat makalah ini dapat selesai tepat pada waktunya. Lalu penulis ucapkan kepada orang tua penulis yang telah mendukung proses perkuliahan agar penulis sukses meraih ilmu yang bermanfaat. Terima kasih juga penulis ucapkan kepada dosen pengajar mata kuliah ini karena mereka yang telah bersedia berbagi dan mengajarkan ilmu untuk mahasiswanya agar bermanfaat bagi nusa dan bangsa. Tidak lupa kepada teman-teman penulis yang telah memberikan semangat kepada penulis untuk berjuang bersama-sama agar lulus di dalam kuliah ini.

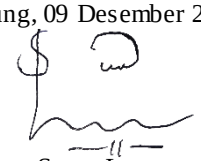
DAFTAR PUSTAKA

[1] Munir, Rinaldi, "Diktat Kuliah IF2211 Strategi Algoritma", Bandung: Institut Teknologi Bandung, 2009.
[2] Cormen, Thomas H, dkk, "Introduction to Algorithms3rdEdition", Massachusetts Institute of Technology, 2009.
[3] "Distributed Computing – Active Projects", URL: http://www.distributedcomputing.info/projects.html diakses tanggal 16 Desember 2013, 19.59 WIB
[4] Miner, Donald, dkk. "MapReduce Design Pattern", California: O'Reilly Media, Inc, 2012
[5] Dean, Jeffrey, dkk. *MapReduce: Simplified Data Processing on Large Clusters*. California: Google, Inc, 2004
[6] "Apache CouchDB", URL: http://couchdb.apache.org/, diakses tanggal 20 Desember 2013, 14.57 WIB

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 09 Desember 2013



Setyo Legowo
(NIM. 13511071)