**UNIVERSITAS GUNADARMA**

**FAKULTAS TEKNOLOGI INDUSTRI**

****

**MapReduce dan Database NoSQL**

**Nama Anggota : 1. Gabriel Neo Gading (52417456)**

**2. Gilang Dwi Praditya (52417543)**

**3. Muhammad Rizky Saputra (54417232)**

**4. Rimadinda Maharani R (55417217)**

**Kelas : 4IA09**

**Nama Dosen : Rheza Andika**

**Mata Kuliah : Komputasi Modern**

**JAKARTA**

**2021**

1. **PENDAHULUAN**
   1. **Latar Belakang**

Database NoSQL dibuat dengan tujuan khusus untuk model data spesifik dan memiliki skema fleksibel untuk membuat aplikasi modern. Database NoSQL dikenal secara luas karena kemudahan pengembangan, fungsionalitas, dan kinerja dalam berbagai skala. Halaman ini menyertakan sumber daya untuk membantu Anda memahami lebih baik database NoSQL dan mulai menggunakannya.

Database NoSQL menggunakan beragam model data untuk mengakses dan mengelola data. Jenis database ini dioptimalkan secara khusus untuk aplikasi yang memerlukan volume data besar, latensi rendah, dan model data fleksibel, yang dicapai dengan mengurangi pembatasan konsistensi data dari database lainnya.

Pertimbangkan contoh pemodelan skema untuk database buku sederhana:

Dalam database relasional, catatan buku sering diselubungkan (atau "dinormalkan") dan disimpan dalam tabel terpisah, dan hubungan ditetapkan dengan pembatasan kunci *primary* dan *foreign*. Dalam contoh ini, tabel Buku memiliki kolom untuk ISBN, Judul Buku, dan Nomor Edisi, tabel Penulis memiliki kolom untuk IDPenulis dan Nama Penulis, dan tabel Penulis-ISBN memiliki kolom IDPenulis dan ISBN. Model relasional didesain untuk mengaktifkan database untuk menegakkan integritas referensial antara tabel di dalam database, dinormalkan untuk mengurangi redundansi, dan umumnya dioptimalkan untuk penyimpanan.

Dalam database NoSQL, catatan buku biasanya disimpan sebagai dokumen JSON. Untuk setiap buku, item, ISBN, Judul Buku, Nomor Edisi, Nama Penulis, dan IDPenulis disimpan sebagai atribut dalam dokumen tunggal. Dalam model ini, data dioptimalkan untuk pengembangan intuitif dan skalabilitas horizontal.

Database NoSQL sangat cocok untuk digunakan dengan berbagai aplikasi modern seperti aplikasi seluler, web, dan gaming yang memerlukan database yang fleksibel, dapat diskalakan, berkinerja tinggi, dan memiliki fungsionalitas tinggi untuk memberikan pengalaman pengguna yang baik.

* Fleksibilitas: Database NoSQL umumnya menyediakan skema fleksibel yang memungkinkan pengembangan yang lebih cepat dan lebih berulang. Model data fleksibel membuat database NoSQL ideal untuk data yang semi terstruktur dan tidak terstruktur.
* Skalabilitas: Database NoSQL umumnya didesain untuk meningkatkan skala dengan menggunakan klaster perangkat keras yang terdistribusi alih-alih meningkatkan skala dengan menambah server yang mahal dan robust. Beberapa penyedia layanan cloud menangani aktivitas di balik operasi ini sebagai layanan yang dikelola sepenuhnya.
* Kinerja tinggi: Database NoSQL dioptimalkan untuk model data spesifik dan pola akses yang memberikan kinerja yang lebih tinggi dibandingkan jika Anda mencoba mendapatkan fungsionalitas yang mirip dengan database relasional.
* Fungsionalitas tinggi: Database NoSQL menyediakan API dan jenis data fungsional yang dibuat secara khusus untuk setiap model data yang sesuai.

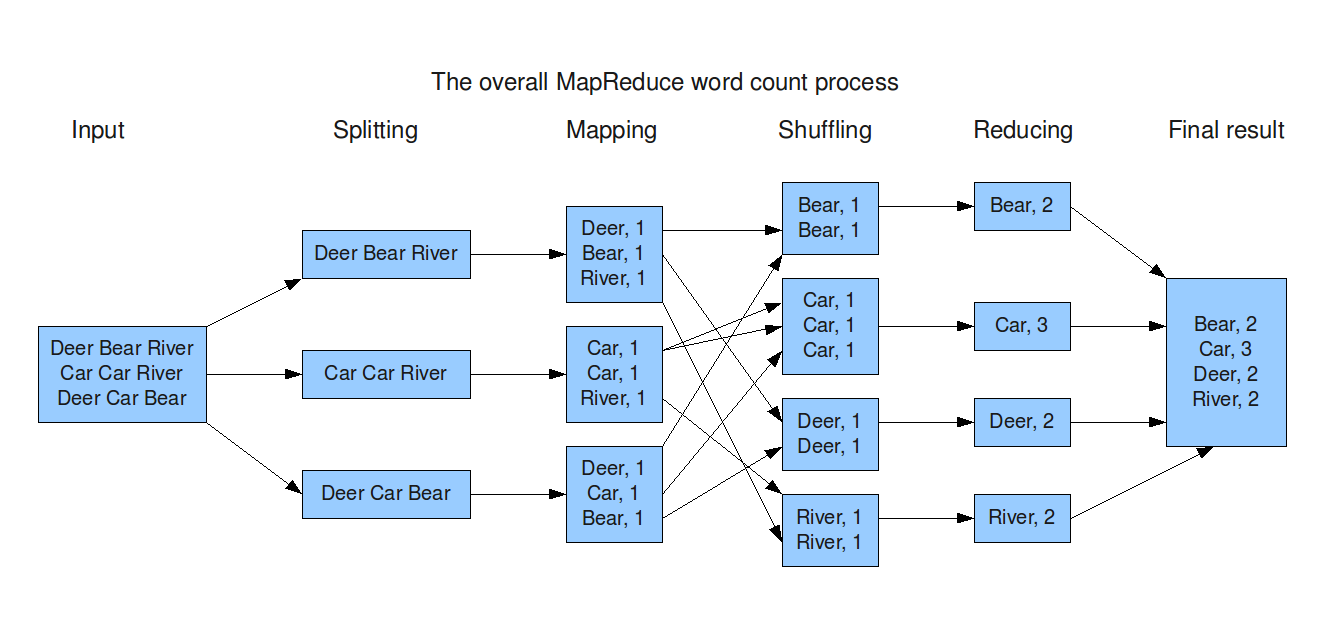
Jenis Database NoSQL :

* Nilai-kunci: Database nilai kunci dapat dipartisi dan memungkinkan pengembangan horizontal pada skala yang tidak dapat dicapai oleh jenis database lain. Kasus penggunaan seperti gaming, teknologi iklan, dan IoT cukup baik dalam menggunakan model data nilai-kunci. Amazon DynamoDB didesain untuk menyediakan latensi satu-digit milidetik untuk setiap skala beban kerja. Kinerja yang konsisten ini merupakan alasan besar fitur Snapchat Stories, yang mencakup beban kerja penulisan ke penyimpanan terbesar dari Snapchat, dipindahkan ke DynamoDB.
* Dokumen: Dalam kode aplikasi, data sering diwakilkan sebagai sebuah objek atau dokumen seperti JSON karena ini merupakan model data yang efisien dan intuitif untuk pengembang. Database dokumen memudahkan pengembang untuk menyimpan dan membuat kueri data dalam database dengan menggunakan format model dokumen yang sama yang mereka gunakan dalam kode aplikasi. Sifat dokumen dan database dokumen yang fleksibel, semiterstruktur, dan hierarkis memungkinkan dokumen dan database dokumen berevolusi sesuai kebutuhan aplikasi. Model dokumen berfungsi baik dengan katalog, profil pengguna, dan sistem manajemen konten di mana setiap dokumen bersifat unik dan berevolusi seiring waktu. Amazon DocumentDB (dengan kompatibilitas MongoDB) dan MongoDB merupakan database dokumen populer yang menyediakan API yang mumpuni dan intuitif untuk pengembangan yang fleksibel dan berulang.
* Grafik: Database grafik bertujuan agar membuat dan menjalankan aplikasi yang berjalan dengan dataset yang selalu terhubung menjadi lebih mudah. Kasus penggunaan umum untuk database grafik antara lain jaringan media sosial, mesin rekomendasi, pendeteksian penipuan, dan grafik pengetahuan. Amazon Neptune merupakan layanan database grafik yang dikelola sepenuhnya. Neptune mendukung model Property Graph dan Resource Description Framework (RDF), memberi pilihan dua API grafik: TinkerPop dan RDF/SPARQL. Database grafik populer termasuk Neo4j dan Giraph.
* Dalam memori: Aplikasi gaming dan teknologi iklan memiliki kasus penggunaan seperti leaderboard, penyimpanan sesi, dan analisis real-time yang memerlukan waktu respons milidetik dan dapat setiap saat memiliki puncak lalu lintas yang besar. Amazon ElastiCache menawarkan Memcached dan Redis, untuk menyediakan beban kerja dengan latensi rendah dan throughput tinggi, seperti McDonald’s, yang tidak dapat dilayani dengan penyimpanan data berbasis disk. Amazon DynamoDB Accelerator (DAX) merupakan contoh lain dari penyimpanan data yang dibuat secara khusus. DAX membuat DynamoDB membaca dan mengurutkan magnitude dengan lebih cepat.
* Pencarian: Beberapa output aplikasi dicatat untuk membantu pengembang untuk memecahkan masalah. Amazon Elasticsearch Service (Amazon ES) dibuat secara khusus untuk memberi visualisasi hampir secara real-time dan analisis data yang dihasilkan mesin dengan membuat indeks, mengumpulkan, dan mencari log dan metrik yang semi terstruktur. Amazon ES juga merupakan mesin pencari yang mumpuni dan berkinerja tinggi untuk kasus penggunaan pencarian teks lengkap. Expedia menggunakan lebih dari 150 domain Amazon ES, 30 TB data, dan 30 miliar dokumen untuk berbagai kasus penggunakan penting, mulai dari pemantauan operasional dan pemecahan masalah hingga pelacakan stack aplikasi terdistribusi dan pengoptimalan harga.



*MapReduce*

Secara definisi, *MapReduce* adalah sebuah model pemograman yang didesain untuk dapat melakukan pemrosesan data dengan jumlah yang sangat besar dengan cara membagi pemrosesan tersebut ke beberapa tugas yang indipenden satu sama lain.



Gambar 1: Alur *MapReduce* Untuk Kasus WordCount. Sumber: <https://www.todaysoftmag.com/images/articles/tsm33/large/a11.png>

Berdasarkan definisi diatas dan Gambar 1, *MapReduce* dapat dibagi menjadi beberapa tahap :

1. **Pemecahan data masukan (*Splitting)*.**Pada proses ini data masukan yang diberikan oleh pengguna *MapReduce* (klien) akan dipecah menjadi bagian-bagian yang lebih kecil. Pada kasus Hadoop *MapReduce*, dalam kondisi ideal, data akan dipecah menjadi beberapa bongkahan berukuran maksimal 128MB.
2. ***Mapping*.***Mapping* adalah salah satu tahap terpenting dari *MapReduce*. Pada fase Mapping, bongkahan data yang telah dipecah akan di proses untuk menghasilkan *intermediary key-value pairs*. Pada contoh *wordcount* (Gambar 1) diatas, data yang mengandung “Dear Bear River” akan diproses sehingga menghasilkan pasangan key-value Dear:1, Bear:1, dan River:1. Dalam fase *Mapping*, bisa jadi ada satu atau lebih mesin pekerja (*worker*) yang melakukan proses terhadap beberapa bongkahan data yang berbeda. Semakin banyak jumlah mesin atau tingkatan parallelisme yang digunakan, maka durasi pemrosesan seluruh data dapat berjalan jauh lebih cepat.
3. **Pengacakan atau *Shuffling***. Fase *mapping* bisa berjalan di satu atau banyak mesin. Akibatnya, pasangan *key-value* yang dihasilkan oleh sebuah *mapper* bisa tersebar di berbagai mesin. Namun, jika pengolahan yang ingin dilakukan adalah perhitungan dengan menggunakan *key* yang sama, maka data dengan *key* yang sama harus berada pada mesin yang sama pada fase *reduce*. Oleh karena itu, sebelum fase *reduce*, fase *shuffling*bertugas untuk mengumpulkan satu atau lebih *key* yang berbeda disebuah mesin tertentu agar aggregasi dapat dilakukan dengan mudah. Pada contoh diatas, seluruh kata ***Bear*** yang dihasilkan fase *mapping*akan berada dalam sebuah mesin yang sama. Begitu juga dengan kata-kata lain.
4. ***Reducing*.**Fase *reducing* bertugas untuk melakukan aggregasi terhadap seluruh pasangan *intermediary* *key-value* dengan *key* yang sama. Pada gambar diatas, pasangan *key-value* Bear:1 dan Bear:1 akan diaggregasi oleh reducer sehingga pada akhirnya *reducer* akan menghasilkan keluaran Bear:2 seperti pada contoh kasus *wordcount*.

**Penggunaan *MapReduce***

Pada umumnya, pengembang aplikasi *MapReduce* akan membutuhkan beberapa hal berikut dalam melakukan analisis data.

1. **Berkas masukan**. Berkas masukan ini dapat berupa berkas-berkas teks yang tersimpan di dalam sebuah media penyimpanan terdistribusi seperti Google *File System* (GFS), Hadoop *File System* (HDFS), AWS S3, Google *Cloud Storage*, dan lain-lain.
2. **Fungsi *Map* & *Reduce*.**Untuk membuat sebuah aplikasi *MapReduce* yang dapat dieksekusi secara paralel (misalkan dengan Hadoop *MapReduce*), pengembang aplikasi menyediakan fungsi khusus yang digunakan untuk melakukan pemrosesan pada fase *map* dan *reduce*. Seluruh hal yang berkaitan dengan penjadwalan, mekanisme penanganan eror, dll. akan dilakukan oleh *MapReduce framework* yang digunakan.

Berikut beberapa contoh fungsi *MapReduce* yang dapat dibuat oleh pengembang piranti lunak :

* *Word Count*
* *Inverted Index*, dll.

Membuat fungsi *map* dan *reduce* bukanlah hal yang sulit. Namun, bagaimana jika kita diminta untuk membuat *framework* *MapReduce* yang digunakan untuk mengendalikan seluruh proses yang terjadi setelah klien memberikan data masukan serta fungsi *map* dan *reduce*?

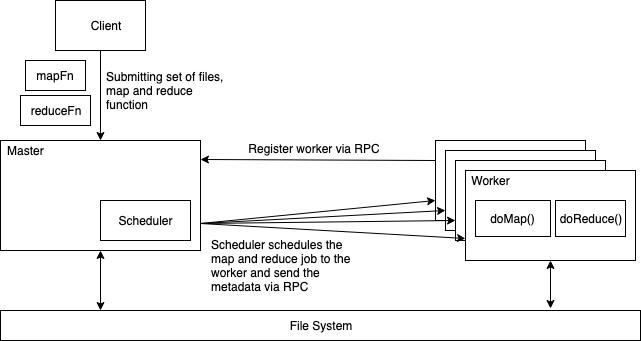
Implementasi *Framework MapReduce* Sederhana :

Sebelum memulai implementasi, untuk membatasi cakupan dari implementasi ini, maka *requirement* dari *framework* *MapReduce* ini adalah sebagai berikut:

1. Implementasi dilakukan dengan menggunakan Golang.
2. Mesin *MapReduce* dapat menerima lokasi berkas masukan di sebuah *file system* serta definisi fungsi *map* dan *reduce* yang diberikan oleh klien.
3. Mesin pekerja dapat melakukan registrasi kepada mesin master agar dapat digunakan ketika klien memulai sebuah tugas *MapReduce*.
4. Mesin master dapat menugaskan eksekusi dari fungsi *map* dan *reduce* kepada mesin pekerja yang telah terdaftar. Apabila terjadi kesalahan dalam pemrosesan karena eror tertentu (seperti eror pada komputasi, atau mesin pekerja tiba-tiba mati), maka mesin master harus dapat mengalokasikan kembali pekerjaan tersebut kepada mesin pekerja lain yang tersedia.
5. Dapat melakukan perhitungan jumlah kata (*word* *count*) dengan menggunakan API *framework* *MapReduce*.

Diluar cakupan:

1. Penggunaan *file system* terdistribusi.
2. Jumlah maksimal pengulangan ketika sebuah tugas gagal dijalankan oleh mesin pekerja. Dan lain-lain.



Gambar 2: Diagram Sistem *Framework MapReduce* Sederhana. Oleh: Imre Nagi

Berdasarkan gambar 2, maka ada beberapa komponen utama yang akan diimplementasikan :

***Scheduler***

*Scheduler* akan menugaskan mesin pekerja untuk melakukan operasi *map* atau *reduce*. Apabila seluruh mesin pekerja sedang melakukan operasi *map* atau *reduce*, maka *scheduler* akan menunggu hingga ada mesin pekerja yang *idle.*Jika terjadi kegagalan pada operasi *map* atau *reduce*, *scheduler* akan berusaha untuk menugaskan mesin pekerja lain untuk melakukan komputasi tersebut.

Berikut adalah antarmuka dari fungsi *scheduler*.

func **schedule**(  
 jobName string,   
 mapFiles []string, //slice of input files  
 nReduce int, //number of reduce task  
 phase jobPhase, //job phase: map or reduce  
 registerChan chan string) //channel of registered workers

*Scheduler* menerima beberapa parameter masukan sebagai berikut:

1. ***jobName***, merupakan nama dari aplikasi *MapReduce* yang akan dieksekusi.
2. ***mapFiles***, adalah kumpulan berkas-berkas yang akan di proses oleh aplikasi *MapReduce*.
3. ***nReduce***, adalah jumlah *reducer* yang dimiliki dalam sebuah *task MapReduce*.
4. ***phase*,**menentukan tipe tugas yang akan dieksekusi oleh mesin pekerja. Masukan untuk parameter ini hanyalah *mapPhase* atau *reducePhase*.
5. ***registerChan*,**merupakan sebuah *go channel* yang menyimpan alamat RPC dari mesin pekerja yang dapat digunakan untuk mengeksekusi sebuah *task map*atau *reduce.*

***Map Controller — doMap()***

*Map Controller /* fungsi *doMap()* bertugas untuk mengelola satu tugas *map*. Fungsi ini akan menggunakan fungsi *mapper* yg didefinisikan oleh klien (*mapF*) untuk melakukan pemrosesan data dan mempartisi keluaran dari fungsi *mapF* menjadi beberapa berkasi terpartisi yang selanjutnya akan digunakan pada fase *reduce*.

Berikut adalah definisi antarmuka dari fungsi *doMap()*:

func **doMap**(  
 jobName string, // the name of the MapReduce job  
 mapTask int, // which map task this is  
 inFile string, // **a** input file which will be processed   
 nReduce int, // num of reduce task for hashing  
 **mapF func(filename string, contents string) []KeyValue**)

*doMap()* memiliki parameter masukan sebagai berikut:

1. ***jobName***, merupakan nama dari aplikasi *MapReduce* yang akan dieksekusi.
2. ***mapTask*,**menyimpan nomor urut dari tugas *map* yang sedang dieksekusi oleh fungsi *doMap()*.
3. ***inFile*,**merupakan lokasi berkas yang akan diproses dengan menggunakan *mapF* yang didefinisikan oleh klien.
4. ***nReduce*,**merupakan jumlah *reducer* yang akan digunakan untuk menentukan berapa banyak partisi yang akan dihasilkan ketika *doMap()* telah dieksekusi.
5. ***mapF***, merupakan sebuah fungsi yang didefinisikan oleh klien. Fungsi ini menerima nama berkas dan konten dari berkas dan menghasilkan keluaran berupa kumpulan *key-value* pair yang selanjutnya akan diproses oleh fungsi *doReduce()*.

Berikut implementasi dari *doMap()*:

Pada implementasi diatas *doMap()* akan mengekstraksi konten dari setiap berkas yang diproses untuk digunakan oleh fungsi *mapF* . Lalu, hasil pemrosesan akan ditulis kembali ke beberapa berkas keluaran yang selanjutnya akan digunakan oleh fungsi *doReduce()*. Perhatikan juga bahwa pada fase ini, proses pengacakan/*shuffling* juga dilakukan dengan menempatkan pasangan *key-value*ke berkas yang telah terpartisi agar proses dapat dilakukan pada satu atau beberapa berkas yang menyimpan *key*yang sama.

Reduce Controller — *doReduce()*

Berikut antarmuka dari fungsi *doReduce()* :

func **doReduce**(  
 jobName string, // the name of the whole MapReduce job   
 reduceTask int, // which reduce task this is  
 outFile string, // write the output here  
 nMap int, // the number of map tasks that were run   
 **reduceF func(key string, values []string) string**)

*Reduce Controller* / fungsi *doReduce()* digunakan untuk mengelola sebuah tugas *reduce*. Fungsi ini akan membaca seluruh berkas *intermediary*yang dihasilkan oleh sejumlah nMap *mapper* dan ditujukan untuk *reducer* tertentu (*reduceTask*).

Berikut adalah penjelasan parameter masukan dari fungsi doReduce() :

1. ***jobName***, merupakan nama dari aplikasi *MapReduce* yang akan dieksekusi.
2. ***reduceTask***, menyimpan nomor urut dari tugas *reduce* yang sedang dieksekusi oleh fungsi *doReduce()*.
3. ***outFile***, merupakan nama berkas yang digunakan untuk menulis keluaran dari *ReduceF*.
4. ***nMap***, merupakan jumlah tugas *map* yang telah dijalankan pada fase *mapping*.
5. ***reduceF***, merupakan fungsi yang didefinisikan oleh klien dan digunakan untuk melakukan aggregasi terhadap seluruh *value* yang dimiliki oleh sebuah *key.*

**Implementasi fungsi *doReduce()***

Pada kode diatas, *doReduce()* akan membaca seluruh berkas *intermediary*yang dihasilkan oleh *doMap().* Untuk setiap berkas, *doReduce()* mengelompokkan seluruh *value*yang memiliki *key* yang sama dalam sebuah data struktur map. Pada akhirnya fungsi *reduceF* dipanggil agar diperoleh hasil aggregat.

***MapF* & *ReduceF* untuk *WordCount***

Contoh *mapF* dan *reduceF* untuk kasus *word* *count*.

Kode diatas merupakan satu-satunya kode yang perlu dipersiapkan oleh klien pengguna *framework MapReduce* yang telah diimplementasikan sebelumnya.

Pada kasus *word count, mapF*memisahkan setiap kata yang terdapat dalam *contents* untuk menghasilkan pasangan *key-value* dengan *key* berupa kata tertentu dengan *value 1*. Fungsi *reduceF* akan menjumlahkan seluruh *value*yang dimiliki oleh sebuah *key.*Sehingga pada akhirnya klien bisa mendapatkan jumlah kemunculan setiap kata yang terdapat pada variabel contets.

*MapReduce Master* dan Pekerja

Implementasi lengkap dari *Master* dan Pekerja (*worker)* dapat dilihat pada *github repository* [*mr-demo*](https://github.com/imrenagi/mr-demo)*.*

Implementasi *Master* dan *Worker* yang digunakan pada tulisan ini merupakan basis kode yangdigunakan pada tugas laboratoriummata kuliah *Distributed System MIT (6.824).* Namun, *implementasi scheduler, doMap* dan *doReduce* merupakan implementasi penulis sendiri.

**Kesimpulan**

*Framework MapReduce* sederhana terdiri dari *Master*, mesin pekerja, *scheduler, map* dan *reduce controller.*