NoSQL: Latar Belakang, Konsep, dan Kritik

Fahri Firdausillah[1], Erwin Yudi Hidayat[2], Ika Novita Dewi[3]
Fakultas Ilmu Komputer, Universitas Dian Nuswantoro, Semarang 50131
[1] fahri@research.dinus.ac.id, [2]erwin@research.dinus.ac.id, [3]ikadewi@research.dinus.ac.id

ABSTRAK

Berkembangnya aplikasi berbasis web yang memerlukan pengolahan data dalam skala besar melahirkan paradigma baru dalam teknologi basis data. Beberapa website seperti Facebook, Twitter, Digg, Google, Amazon, dan SourceForge menyimpan dan mengolah data puluhan giga setiap harinya, dan total keseluruhan data yang disimpan oleh applikasi tersebut sudah mencapai ukuran petabyte. Ukuran data yang sangat besar menimbulkan permasalahan dari segi skalabilitas, karena pertambahan data yang terjadi setiap saat. Peningkatan kemampuan server secara vertikal yang dimiliki Relational Database Management System (RDBMS) terbatas pada penambahan prosesor, memori, dan media penyimpanan dalam satu node server yang terbatas. Sedangkan peningkatan kemampuan server secara horizontal yang meliputi penambahan perangkat server baru dalam suatu jaringan memerlukan biaya yang mahal dan sulit dalam pengelolaannya. Salah satu cara yang diterapkan oleh website berskala besar untuk mengatasi permasalahan tersebut adalah dengan menggunakan NoSQL, sebuah paradigma basis data yang merelaksasikan aturan-aturan konsistensi yang terdapat pada basis data relasional. Jika RDBMS menggunakan aturan Atomicity, Consistency, Isolation, dan Durability (ACID) untuk penyimpanan dan pengolahan data, maka NoSQL menggunakan paradigma Basically Available, Soft State, and Eventually consistent (BASE) untuk merelaksasikan aturan tersebut. Hasilnya, NoSQL dapat mengolah data dalam jumlah besar dengan memartisi data ke dalam beberapa server secara lebih mudah. Makalah ini membahas dan menjelaskan latar belakang kemunculan, konsep dasar, dan penggunaan NoSQL.

Kata kunci : *Basis data, RDBMS, Skalabilitas, NoSQL*

1. PENDAHULUAN

RDBMS merupakan suatu sistem khusus yang mengatur organisasi, penyimpanan, akses, keamanan dan integritas data. *RDBMS* merupakan sistem pengelolaan basis data yang paling banyak digunakan saat ini, meskipun terdapat berbagai macam sistem yang lain. *RDBMS* menyimpan informasi dalam suatu kumpulan tabel. Suatu desain *RDBMS* dikatakan bagus jika terdapat normalisasi basis data didalamnya.

Beberapa aplikasi seperti *weblog* dan perbankan menggunakan *Structure Query Language (SQL)* dan *RDBMS* menawarkan solusi yang baik dalam penggunaan basis data. *RDBMS* menerapkan *Atomicity, Consistency, Isolation, and Durability (ACID)* dalam penyimpanan dan pengelolaan data [1]. *Atomicity* mengharuskan suatu modifikasi basis data mengikuti aturan "*all or nothing*". Suatu transaksi dikatakan *atomic* indikasinya adalah jika satu transaksi gagal maka menyebabkan kegagalan keseluruhan transaksi. *Consistency* mengatur validitas data yang akan dituliskan dalam basis data. Jika suatu transaksi berhasil dijalankan maka akan dilakukan pengambilan basis data dari suatu kondisi yang konsisten dengan aturan-aturannya menuju kondisi lain yang juga konsisten terhadap aturan-aturan yang berlaku. *Isolation* menyaratkan bahwa transaksi yang berlipat ganda dalam satu waktu tidak berpengaruh terhadap pengeksekusian transaksi yang lain. *Durability* memastikan bahwa transaksi yang dilakukan tidak akan hilang dengan melakukan *backup* basis data dan *log* transaksi yang memberikan fasilitas restorasi dari transaksi yang dilakukan walaupun kemudian banyak terjadi kegagalan pada *software* dan *hardware*nya.

Kombinasi penggunaan *RDBMS*, *SQL*, dan *ACID* memberikan infrastruktur pemrograman yang baik untuk aplikasi perbankan, toko *online*, dan aplikasi-aplikasi lain yang menitik beratkan pada manajemen transaksi. Tetapi tidak semua aplikasi memerlukan penggunaan *ACID* karena konsekwensi yang harus ditanggung dalam kinerja aplikasi dan fitur-fiturnya.

Generasi terbaru dari perangkat lunak internet sangat tergantung pada penggunaan sistem basis data. Kenyataanya untuk meningkatkan kemampuan dan kecepatan operasi, pengembang dapat merelaksasikan beberapa aturan ketat yang ada pada *RDBMS* seperti *consistency* dan *atomicity*. Sebagai realisasinya jenis basis data baru telah diperkenalkan beberapa tahun lalu yang dikenal dengan *NoSQL* (singkatan dari *Not-Only-SQL*). Beberapa aplikasi berbasis web yang telah menerapkan penggunaan *NoSQL* diantaranya adalah *Google* dengan *BigTable*, *Amazon* dengan *Dynamo*, dan *Facebook* dengan *Cassandra* dan *Hadoop*. Dengan mengesampingkan aspek *ACID*, *NoSQL* menerapkan konsep *BASE* (*Basically Available, Soft state*, dan *Eventually consistent*) untuk meningkatkan *availability* dan *partitioning* data. *Basically Available* memastikan bahwa sistem bekerja sepanjang waktu. *Soft state* merupakan suatu kondisi dimana sistem tidak harus konsisten setiap saat, dan *Eventually consisten* yang menekanakan bahwa sistem akan menjadi konsisten beberapa waktu

kemudian. Makalah ini menjelaskan tentang permasalahan pada RDBMS yang melatar belakangi kemunculan *NoSQL*, konsep umum NoSQL, dan juga beberapa kritik terhadap konsep NoSQL. Karena NoSQL bukanlah Peluru Emas yang dapat menyelesaikan semua masalah basis data, makalah ini dapat memberikan gambaran kapan seseorang/instansi perlu menggunakan NoSQL, dan kapan harus menggunakan RDBMS.

2. PERMASALAHAN DALAM RDBMS

Beberapa permasalah yang muncul dalam *RDBMS* biasanya berhubungan dengan kinerja dan skalabilitas. Misalnya *DBMS* komersial yang ada sekarang hanya beberapa saja yang menawarkan atau bahkan tidak ada yang menawarkan fitur untuk mengatasi masalah data *geoscience* dan data *environmental* terutama untuk jenis data yang tidak lengkap dan tidak tepat [4].

Dalam relasi basis data untuk jumlah data yang sangat besar sangat diperlukan optimisasi *query*. Berbeda dengan generasi navigasi basis data sebelumnya, sebuah *query* dalam relasi basis data menentukan data apa yang akan diperoleh dari basis data bukan bagaimana cara mendapatkannya [5]. Untuk mempercepat perolehan data yang diperlukan *RDBMS* menggunakan kunci primer dan kunci sekunder serta penggunaan *index*. Data yang selalu bertambah akan diikuti dengan pertumbuhan masalah didalamnya [6]. Misalnya penggunaan *query* yang tidak tepat dapat menghambat kinerja basis data yang sudah ada. Suatu *disk* pembaca tunggal akan memerlukan waktu yang lama dalam pencarian ribuan tabel [7]. Mengoptimalkan *query* untuk mengurangi penggunaan sumberdaya secara keseluruhan sangat penting untuk diterapkan karena akan meningkatkan kinerja *order of magnitude* secara keseluruhan.

Dalam suatu relasi basis data, misalnya terdapat *query* student_id untuk siswa yang akan mengikuti kelas tari. Hubungan antara siswa, tarian, dan kelas tari tidak dapat disimpulkan karena sistem hanya akan menjalankan *query* yang menggunakan kata kunci siswa, tarian dan kelas tari [8]. Kunci dari desain basis data yang baik adalah keakurasian dalam menentukan kardinalitas suatu relasi. Jika suatu kardinalitas relasi tidak dimodelkan dengan tepat maka akan terjadi duplikasi, redundansi dan anomali data [9]. Sistem basis data tradional menyediakan bahasa *query* dengan kualitas yang ekspresif, tetapi penerepan *query* ini dalam tingkatan teratas sistem yang berbasis *file* tidaklah mudah dilakukan.

Basis data relasional dapat ditingkatkan kinerjanya dengan meningkatkan kinerja komponen-komponen komputer seperti meningkatkan kemampuan *processor*, menambahkan modul *Random Memory Access (RAM)*, dan memberikan media penyimpanan tambahan. Namun peningkatan tersebut hanya dapat dilakukan pada sebuah *node server*, jika kemampuan suatu *node* sudah mencapai batas maksimum spesifikasi maka untuk meningkatkan kemampuan harus membeli perangkat server baru yang lebih mahal dengan spesifikasi yang lebih tinggi. Salah satu cara lain untuk meningkatkan kemampuan server adalah dengan menambahkan perangkat baru pada jaringan dan mempartisi basis data agar penyimpanannya dapat dibagi pada server lama dan server tambahan. Cara tersebut relatif lebih murah namun tidak mudah untuk diimplementasikan pada RDBMS karena aturan konsistensi data yang lebih diprioritaskan, perlu perekayasaan dengan pemrograman yang berakibat pada tingginya biaya implementasi.

3. KONSEP DASAR NoSQL

Bagian ini akan menjelaskan fitur yang dimiliki oleh basis data *NoSQL* terutama dalam hal distribusi data dan *query* dalam beberapa server yang sama. Telah banyak produk basis data *NoSQL* dengan berbagai keunggulannya masing-masing. Beberapa diantaranya adalah *CouchDB* yang memiliki nilai konkruensi dan *RESTfull HTTP request*, *simpleDB* yang memiliki kesederhanaan dan fleksibilitas dalam pemeliharaannya, dan *Google BigTable* yang memiliki kuota yang terbatas [12].

Eric brewer [2] menyatakan bahwa *NoSQL* didasarkan pada teori *CAP* yaitu pemilihan dua dari tiga aspek yang ada yang harus dipenuhi oleh basis data yaitu *Consistency*, *Avaibility*, dan *Partition-Tolerance*. (1) *Consistency Avaibility* (*CA*) berseberangan dengan *Partition-Tolerance* dan biasanya berhubungan dengan replikasi. (2) *Consistency Partition-Tolerance* (*CP*) berseberangan dengan *Avaibility* dalam penyimpanan data. (3) *Avaibility Partition-Tolerance* sistem mencapai kondisi *eventual consistency* [3] melalui replikasi dan verifikasi yang konsisten dalam *node* yang telah terbagi-bagi. Dalam basis data *NoSQL* penerapan konsep tersebut diterjemahkan dalam empat konsep dasar yaitu *Non-Relational*, *MapReduce*, *Schema Free*, dan *Horizontal Scaling*.

3.1 Non-Relational

Konsep *Non-Relational* dalam basis data *NoSQL* meliputi hirarki, graf, dan basis data berorientasi obyek yang sudah terlebih dahulu ada sejak tahun 1960 sebelum akhirnya basis data relasional muncul pada tahun 1970 [13]. Penggunaan basis data non-relational kembali merebak seiring dengan bertambahnya aplikasi berbasis web yang memerlukan banyak penyimpanan data. Meskipun memiliki kelemahan pada konsistensi dan redundansi data, namun basis data non-relasional dapat menyelesaikan beberapa permasalahan terkait dengan *avaibility*, dan *partition-*

tolerance. Tugas pengecekan konsistensi dan redundansi data diserahkan pada sisi aplikasi, sedangkan basis data non-relational hanya bertugas memanipulasi penyimpanan saja.

3.2 MapReduce

MapReduce merupakan model pemrograman yang diadaptasi dari pemrograman fungsional yang diimplementasikan untuk mengolah *dataset* yang sangat besar. Tujuan dari *MapReduce* adalah merancang suatu abstraksi baru yang memungkinkan pengguna untuk membuat antarmuka pemrograman sederhana dan menyembunyikan detail yang rumit dari pararelisasi, *fault-tolerance*, distribusi data, dan *load balancing* dalam pustaka pemrogramannya. Hasilnya menunjukkan bahwa penerapan *MapReduce* dapat menyederhanakan antarmuka pemrograman yang dapat mendukung paralelisasi dan distribusi komputasi skala besar secara otomatis [14].

Pemrograman dengan *MapReduce* telah sukses diterapkan penggunaannya oleh *Google* untuk berbagai tujuan, salah satunya adalah *Google indexing*. Dalam kinerjanya, *Google* menerapakan ribuan mesin yang bekerja pada ratusan *terabytes* data dengan lokasi server yang tersebar di beberapa lokasi. Jenis arsitektur seperti ini dapat mengurangi waktu kinerja yang diperlukan. Pembangunan arsitektur *Google* dengan menggunakan *MapReduce* memerlukan waktu hanya dalam beberapa jam saja dibandingkan dengan tidak menerapkan *MapReduce* yang memerlukan waktu selama berbulan bulan. Penggunaan *library* dalam *MapReduce* memiliki beberapa keuntungan seperti *load balancing*, optimasi perangkat penyimpanan yang nantinya akan meningkatkan keefisiensian sistem dan kemudahan dalam penggunaannya. Mudahnya, *MapReduce* bekerja dengan membagi proses menjadi dua fase, yaitu tahap *map* dan tahap *reduce*. Seorang *programmer* dapat memanfaatkan dua fungsi ini bersama fungsi *key-value pairs* sebagai *input* dan *output* untuk mencapai semua fase [15].

Berikut ini merupakan contoh penggunaan fungsi *Map()* dan *Reduce()* yang diambil dari makalah Ricky Ho[16]. Tujuan dari penggunaan *MapReduce* dalam kasus ini adalah menghitung jumlah *distinct path* diantara orang-orang yang memiliki koneksi dan akan disarankan ke orang dalam suatu situs pertemanan. Fungsi *MapReduce* digunakan untuk mencari 10 orang teratas yang saling terkoneksi.

Untuk fungsi *Map()*, sebuah *Cartesian Product* dikerjakan untuk semua pasangan teman. Kita juga perlu menghilangkan pasangan yang terkoneksi langsung. Oleh karena itu fungsi *Map()* juga harus memunculkan pasangan orang yang terkoneksi langsung. Kita perlu mengambil *key space* sedemikian rupa sehingga semua *key* dengan pasangan yang sama akan mempunyai pengurangan yang sama. Disisi lain, kita memerlukan pasangan yang berhubungan langsung sebelum pasangan yang mempunyai pemisah.

Pada Fungsi *Reduce()* semua *key* yang mencapai *reducer* yang sama akan diurutkan. Jadi pasangan yang terkoneksi langsung akan ada sebelum pasangan yang tidak terkoneksi. *Reducer* hanya perlu untuk mengecek jika pasangan pertama merupakan koneksi langsung dan jika iya, maka lanjutnya sisanya.

```
Input record ...
person -> connection_list e.g. "ricky" => ["jay", "john", "mitch", "peter"]
also the connection list is sorted by alphabetical order

def map(person, connection_list)
loops
  for each friend1 in connection_list
already
     emit([person, friend1, 0])
     for each friend2 > friend1 in
connection_list
         emit([friend1, friend2, 1],  1)

def partition(key)
choose a reducer
  return super.partition([key[0], key[1]])
def reduce(person_pair, frequency_list)
  if @current_pair != [person_pair[0],
person_pair[1]]
      @current_pair = [person_pair[0],
person_pair[1]]
person
      @skip = true if person_pair[2] == 0
  if !skip
      path_count = 0
      for each count in frequency_list
```

Gambar 1: Fungsi Map() dan Reduce()

3.3 Schema-Free

NoSQL dan *RDBMS* mempunyai perbedaan dalam hal penerapan skema basis data. Dalam basis data relasional, sebuah tabel didesain dengan peraturan skema yang ketat. *NoSQL* menyimpan data dengan aturan yang lebih longgar, artinya tidak seperti basis data yang berdasarkan *SQL* tradisional , *NoSQL* tidak memiliki tabel, kolom, *primary* dan *foreign key*, join, dan relasi [17]. Dalam pengembangan basis data relasional, *developer/database administrator* harus berhati-hati dalam menentukan bagaimana tabel saling berelasi dan *field* yang ada didalam setiap tabel. Karena perubahan skema dalam RDBMS dapat menimbulkan masalah ketergantungan dan integritas, seperti timbulnya kolom *null* dan relasi kunci yang tidak cocok. Hal ini bukan masalah dalam *NoSQL* karena adanya penerapan *schema-free*. Setiap dokumen bertanggung jawab terhadap isinya sendiri, maksutnya *null value* dapat dihilangkan dalam beberapa baris, dan *field* baru dapat didefinisikan dalam setiap dokumen secara indpenden [17]. Salah satu produk basis data *NoSQL* yang menerapkan fitur *schema-free* adalah *Cassandra*. Dalam *Cassandra*, pengembang hanya perlu mendefinisikan *keyspace* sebagai pertimbangan *container* dan *contains column families*. *Keyspace* hanya digunakan sebagai *logical namespace* yang dapat dimasukkan dalam *configurations* dan *hold column families* [18]. *Column families* artinya sebuah nama untuk relasi data dan hirarki kolom (memungkinkan penyisipin tabel dalam kolom). Disamping itu, kita hanya perlu menambahkan data dalam tabel, menggunakan kolom, tanpa menentukan kolomnya terlebih dahulu. Dalam *Cassandra* kita hanya perlu mendefinisikan tabel *namespace* dan hirarki kolom didalamnya, memodelkan dan mendesain tabel join adalah masalah dalam basis data relasional bukan NoSQL.

Manfaat lain dalam penggunaan *schema-free* adalah penghematan dalam media penyimpanan. Dalam basis data relasional, setiap *field* yang ada dalam tabel harus mempunyai nilai, walaupun nilai itu *null*. Model data *schema free* artinya setiap baris memungkinkan memiliki nilai sebanyak yang telah didefinisikan dalam tiap *fields*, dan tidak perlu menggunakan nilai yang memang tidak diperlukan [18]. Kelemahan dalam penggunaan *schema free* adalah memunculkan lemahnya pendefinisian struktur yang menungkinkan terjadinya penggunaan basis data yang tidak konsisten. Jika tujuan pembangunan basis data didasarkan pada konsistenti yang ketat, seperti *wiki*, *document management systems*, *discussion forums*, *blogs*, dan *support management systems*, basis data relasional masih merupakan pilihan yang tepat [17].

3.4 Horizontal Scaling

Horizontal scaling memungkinkan basis data dijalankan pada beberapa server untuk meningkatkan kemampuan perangkat penyimpanan dan meningkatkan efisiensi waktu [19]. Hal ini memerlukan kemampuan dinamis pemartisian data dalam serangkaian *node* (seperti *storage hosts*) dalam suatu *cluster server*. Kemampuan untuk meningkatkan kemampuan dengan menambahkan beberapa komputer sangatlah penting dilakukan untuk data yang jumlahnya banyak, karena *vertical scaling* dilakukan dengan meningkatkan kemampuan spesifikasi *single server* (misalnya penambahan prosesor, memori, dan peralatan penyimpanan) terbatas untuk dilakukan dan lebih mahal [13].
Horizontal scaling berarti memungkinkan dilakukannya penambahan server dalam satu jaringan dan *user* tidak sadar jika ada *hardware* yang diganti dari sisi server (*transparent*). Ada beberapa teknik partisi yang digunakan dalam basis data untuk melakukan *horizontal scaling*, salah satunya adalah *consistent hashing* yang digunakan dalam *Cassandra* dan *Amazon Dynamo* [19]. Kunci dari menerapkan *consistent hashing* adalah membuat suatu lingkaran atau "*ring*". Setiap *node* dalam sistem yang ditandai dengan *random value* dalam suatu *space* yang merepresentasikan posisi dari *ring*. Suatu kunci *item* ditandai dengan sebuah *node* untuk memperoleh posisinya pada *ring*, kemudian berpindah menuju *node* selanjutya yang sudah ditandai. *Node* memerankan peranan penting sebagai koordinator untuk kunci yang akan digunakan dalam *route request*. Kemudian, setiap *node* menjadi tanggung jawab dari daerah yang ada di *ring* diantara *noder* dan *node* dalam *ring* sebelumnya. Pejelasan detail mengenal *Dynamo* dijelaskan oleh DeCandia dkk [20].

4. MODEL DATA RELASIONAL NOSQL

NoSQL merupakan basis data non relasional dengan *schema-free* yang memunculkan pertanyaan bagaimana *NoSQL* dapat melakukan partisi untuk data yang berukuran besar, melakukan *query*, replikasi data, dan mendukung adanya konsistensi. Bagian ini berisi penjelasan empat model data *NoSQL*, yaitu *column-oriented, document-oriented, object-oriented* dan *graph-oriented* [21].

4.1 Column-oriented

Penerepan *column-oriented* terdapat dalam *Casssandra*. *Cassandra* menggunakan distribusi multidimensional *map indexed* dengan sebuah *key*. Baris kunci yang sering digunakan dalam *Cassandra* adalah *string* dengan panjang 16-36 *bytes*. Setiap kolom digabungkan menjadi sebuah *coloumn families*. Wei Kang [21] menerapkan beberapa konsep dalam penerapan *column-oriented*. Suatu kolom unit *atomic* dari informasi didukung oleh *Cassandra* yang diekspresikan dengan nama *value*. *Super-column* merupakan gabungan dari kolom dengan nama yang umum dan digunakan untuk pemodelan tipe data yang komplek. Baris secara unik mengidentifikasikan data yang terdapat dalam

column dan *super-column*. Dalam *Cassandra* baris dapat dikenali dengan sebuah kunci. *Column Family* merupakan bagian dari suatu unit abstraksi yang berisi baris kunci yang tergabung dalam *column* dan *super column* yang memiliki struktur data yang tinggi. *Keyspace* merupakan level teritnggi dari unit informasi yang terdapat dalam *Cassandra*. Kumpulan *column families* sebenarnya merupakan subordinat dari satu *keyspace*. Pada intinya model data *column-oriented* memungkinkan suatu aplikasi secara bebas untuk mengembangkan bagaimana informasi disusun berdasarkan suatu desain *schema*.

4.2 Document-oriented

Contoh penerapan model data *document-oriented* terdapat dalam *CouchDB* yang dibangun oleh *IBM*^{TM}. Basis data yang memiliki model data *document-oriented* sangat bermanfaat untuk suatu domain yang bentuk masukannya dokumen yang tidak terstruktur seperti *web pages, wikis, discussion forums,* dan *blogs*. Data tersimpan dalam basis data *CouchDB* yang mencakup serangkaian dokumentasi yang berisi beberapa atribut dan nilai dengan masing-masing *id* yang unik dan *metadata*. *CouchDB* tidak pernah melakukan *overwrite document*, melainkan menambahkan dokumen baru ke basis data bila diperlukan seperti ketika terjadi proses *update* [21].

4.3 Object-oriented

Basis data berorientasi objek adalah model basis data dimana informasi direpresentasikan dalam bentuk objek yang digunakan dalam pemrograman berorientasi obyek. *OrientDB* adalah contoh dari basis data *object-oriented*. Dokumen *OrientDB* yang ada dalam suatu *cluster*, dapat berupa fisik, logis atau *in-memory*, yang digunakan untuk menyimpan *link* ke dalam data. *Cluster* adalah cara yang sangat umum untuk mengelompokkan *record*, hal ini merupakan suatu konsep yang tidak ada dalam basis data relasional. Cara ini dapat mengelompokkan semua *record* pada jenis tertentu, atau dengan nilai-nilai tertentu. *OrientDB* menggunakan segmen data untuk menyimpan isi *record*. Segmen data mirip dengan *file physical cluster* yang menggunakan dua atau lebih *file*, yaitu satu atau beberapa *file* dengan ekstensi "*oda*" (*Orient Data*) dan hanya satu *file* dengan ekstensi "*odh*" (*Orient data Holes*) [21].

4.4 Graph-oriented

Basis data grafik (*GraphDB*) adalah basis data yang menggunakan struktur grafik yang berisi *node*, *edge*, dan properti untuk mewakili dan menyimpan informasi. *GraphDB* diperlukan untuk data grafik yang berskala besar, terutama yang dipergunakan oleh para peneliti biologi jaringan dan situs jaringan sosial, seperti *Facebook*, dan *Twitter*. *GraphDB* memetakan secara langsung objek ke aplikasi dan lebih intuitif untuk menggambarkan *data set* asosiatif. Beberapa keuntungan dari *GraphDB* adalah [21]: *Intuitive*, dimengerti oleh pikiran manusia, yaitu menggambarkan entitas dan hubungan sebagai grafik masalah umum yang akrab dengan manusia; *Elemental* untuk ilmu komputer, yaitu grafik, terutama grafik pohon (seperti *binary-tree*, *B+ tree, red-black tree*) berfungsi sebagai struktur data dasar dalam ilmu komputer dan berbagai masalah (*shortest path* dan *max-flow*) dapat diubah dan diselesaikan dengan algoritma grafik; *Ubiquitous*, yaitu pemodelan *ER* ke model jejaring sosial selalu dikelilingi oleh grafik baik di komputer ataupun dalam kenyataan.

Keunggulan lain dalam *GraphDB* adalah biasanya lintasan grafik digunakan sebagai pengganti operasi join yang berpengaruh dalam efisiensi *query* [22]. *GraphDB* juga tergantung pada kurangnya *schema* yang kaku di mana suatu *schema* dapat selalu diubah dengan mudah pada grafik, karena struktur grafik sendiri cukup fleksibel untuk mewakili perubahan melalui edit *edge* dan properti. *GraphDB* juga dapat mendukung semua fitur basis data yang kuat [23].

5. KRITIK NoSQL

Brewer membuat teorema basis data yang menjelaskan bahwa sebuah DBMS hanya dapat memilih paling banyak dua diantara tiga karakteristik ada, yaitu konsistensi data, ketersediaan data, dan toleransi untuk dipecah menjadi beberapa partisi [24]. *RDBMS* memilih untuk memprioritaskan konsistensi data dan ketersediaan data, sedangkan mayoritas *NoSQL* memilih untuk menggunakan ketersediaan data dan toleransi pemartisian data, karena kebanyakan basis data *NoSQL* digunakan pada aplikasi yang harus menyimpan dan memproses banyak data dengan cepat seperti website jejaring sosial, mesin pencari, dan lain sebagainya. Sebagai konsekuensi dari pilihan tersebut, *NoSQL* mengorbankan kemudahan untuk konsistensi data. Ketidak konsistenan sebuah basis data dapat menimbulkan kesalahan yang fatal dalam beberapa kasus. Sebagai contoh dalam aplikasi penjualan *online* seperti pada *Amazon*, saat ada dua atau lebih pembeli yang melakukan transaksi pembelian pada barang yang sama, pada sistem basis data yang tidak konsisten akan mengijinkan kedua transaksi tersebut. Akibatnya, apabila stok barang yang bersangkutan hanya satu, maka salah satu dari pembeli akan dirugikan karena validasi transaksi dan pembayaran tetap berhasil, namun barang tidak tersedia.

Meskipun aplikasi yang menggunakan *NoSQL* dapat merekayasa validasi konsistensi data melalui pemrograman, Xiang [25] menawarkan solusi dengan membuat *middleware layer* yang menggunakan algoritma *hash* untuk mempartisi dan mereplikasi data. Algoritma yang digunakan dapat memastikan konsistensi data pada basis data *NoSQL* yang terpartisi pada beberapa *node*, bahkan juga mampu mengidentifikasi jika ada salah satu *node server* yang hilang (rusak atau tidak

terhubung). Sayangnya solusi semacam ini juga mengakibatnya menurunnya performa *server* dan meningkatnya waktu pemrosesan *query*.

Lebih jauh lagi Dewit [26] menuliskan beberapa kritikan untuk konsep NoSQL. Ada lima poin yang disoroti oleh Dewit dan dianggap sebagai kegagalan NoSQL:

1. *MapReduce* merupakan langkah mundur dalam pemrosesan data. Basis data relasional dilahirkan untuk mengurangi beberapa kelemahan yang dimiliki *flat file*, namun konsep NoSQL mempunyai karakteristik yang hampir sama dengan *flat file* termasuk beberapa kelemahan yang sudah diselesaikan oleh basis data relasional.
2. *NoSQL* merupakan sebuah implementasi sub-optimal, karena menggunakan paradigma *brute force* untuk pemrosesan data, bukan menggunakan *indexing* yang lebih baik.
3. Konsep *NoSQL* sama sekali bukanlah sesuatu yang baru. Konsep tersebut sudah dikenal 20an tahun yang lalu, dan sekarang hanya dikemas menggunakan cara baru.
4. Tidak menyediakan fitur yang sudah banyak digunakan oleh pengembang yang menggunakan basis data relasional.
5. Tidak compatible dengan semua perangkat-perangkat *DBMS* yang sudah biasa digunakan oleh pengguna.

Meski demikian, kritikan tersebut juga mendapat sanggahan dari beberapa praktisi seperti Mark C dkk [27] dan [28]. Para penyanggah menekankan bahwa pembandingan *NoSQL* dengan basis data relasional adalah tidak tepat, karena keduanya memiliki tujuan penggunaan yang berbeda, basis data relasional untuk aplikasi yang sensitif terhadap konsistensi sedangkan *NoSQL* ditujukan untuk pemrosesan data skala besar dan tidak sensitif terhadap konsistensi. *NoSQL* juga memang bukan untuk menawarkan solusi yang baru, namun lebih pada solusi lama yang dapat digunakan untuk menangani permasalahn baru.

6. KESIMPULAN

Konsep utama dari penerapan *NoSQL* adalah bagaimana mengatasi jumlah data yang sangat besar dan ledakan data dalam aplikasi web sebagai paridigma baru dalam penerapan teknologi basis data. Permasalahan dalam *RDBMS* mengenai skalabilitas dan partisi data dapat diatasi dengan menggunakan *NoSQL*. *NoSQL* memiliki beberapa keunggulan seperti basis data non-relasional (meliputi hirarki, graf, dan basis data *object oriented*); *MapReduce* yang diambil dari fungsi pemrograman diterapkan untuk menghasilkan *dataset* yang besar; *Schema-free* yang memungkinkan dimana tidak terdapat tabel, kolom, kunci primer dan sekunder, join, dan relasi; *Horizontal scaling* yang memungkinkan basis data untuk dijalankan dibeberapa *server* untuk meningkatkan penyimpanan dan peningkatan waktu akses untuk mengatasi permasalahan banyaknya data. Empat model data *NoSQL* juga telah berhasil diinvestigasi dalam penelitian ini, yaitu *column-oriented, document-oriented, object-oriented,* dan *graph-oriented*. Sistem pembagian data dapat dilakukan dengan memenuhi dua dari tiga teori *CAP*.

Untuk peningkatan *horizontal scaling, NoSQL* mengorbankan konsistensi. Meskipun demikian *NoSQL* merupakan alternatif dari *RDBMS* dalam hal pendistribusian data, bukan penanganan masalah secara keseluruhan terutama transaksi yang tinggi. *NoSQL* tidak menerapkan konsistensi dan integritas data, hal ini membuat *programmer* harus bekerja ekstra dalam untuk mengatasinya dari sisis pemrograman.

DAFTAR PUSTAKA

[1] Ran Tavory: Introduction to NoSQL and Cassandra, Part 1. http://prettyprint.me/2010/01/09/introduction-to-nosql-and-cassandra-part-1, diakses pada 16 February 2011
[2] Julian Browne Brewer's CAP Theorem http://www.julianbrowne.com/article/viewer/brewers-cap-theorem, diakses pada 30 January 2011
[3] Eventual Consistency http://books.couchdb.org/relax/intro/eventual-consistency, diakses pada 10 February 2011.
[4] S. Henley, "The Man Who wasn't There: The Problem of Partially Missing Data," Computers & Geosciences, vol. 31, Jul. 2005, pp. 780-785.
[5] A. Ghazal and R. Bhashyam, "Dynamic Constraints Derivation and Maintenance in the Teradata *RDBMS*," International Conference on Database and Expert Systems Applications, 2001, pp. 390-399.
[6] N. Arif, NoSQL: the End of RDBMS? http://arifn.web.id/blog/2010/05/05/nosql-the-end-of-*RDBMS*.html, diakses pada 15 February 2011
[7] O. Erling and I. Mikhailov, "Virtuoso: RDF Support in a Native *RDBMS*," Semantic Web Information Management, Springer-Verlag Berlin Heidelberg, 2010, pp. 501-519.
[8] K.J. Lee and T.K. Whangbo, "Semantic Mapping between *RDBMS* and Domain Ontology," IEEE, 2007, pp. 1007-1012.
[9] S.R. Alapati, "Expert Oracle Basis data 11 g Administration," Communications, 2009, pp. 19-41.
[10] J. Chmura, N. Ratprasartporn, and G. Ozsoyoglu, "Scalability of Database for Digital Libraries," International Conference on Asian Digital Libraries, 2005, pp. 435 - 445.
[11] M. Stonebraker. SQL Database versus No-SQL Database. Communications of the ACM, Volume 53, Issue 4. 2010.
[12] Uma Bhat and Shraddha Jhadav, Moving Towards Non-Relational Databases, International Journal of Computer Applications, 2010.
[13] N. Leavitt. Will NO-SQL Basis datas Live Up to Their Promise? IEEE Computer Society, Volume 43, Issue 2, 2010.

[14] Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, Proceeding of OSDI, 2004.
[15] Tom White, Hadoop The Definitive Guide, O' Reily, 2010
[16] Ricky Ho, Map/Reduce to recommend people connection, http://horicky.blogspot.com/2010/08/mapreduce-to-recommend-people.html, diakses pada 13 February 2011.
[17] Joe Lennon, Beginning CouchDB, New York, Apress 2009
[18] Eben Hewitt, Cassandra: The Definitive Guide, United States of America, O'Reilly Media, Inc 2010.
[19] P. Malik A. Lakshman. Cassandra - a decentralized structured storage system. The 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware (LADIS 09), October 2009.
[20] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in SOSP '07. New York, NY, USA: ACM, 2007, pp. 205–220.
[21] Wei, Kang, Sicong, Tan, Qian, Xiao, Amiri, Hadi, "An Investigation of No-SQL Data Stores". http://www.comp.nus.edu.sg/~ozsu/cs5225/Projects/CS5225Final_v15.pdf, diakses pada 10 February 2011.
[22] Hyper graph DB, http://www.hypergraphdb.org/index, diakses pada February 15, 2011
[23] The Web Graph Database, http://infogrid.org/, diakses pada February 16, 2011
[24] N. Lynch and S. Gilbert, "Brewer's Conjecture and The Feasibility of Consistent, Available, Partition-tolerant Web Services", ACM SIGACT News, 2002, vol. 33 Issue 2, pp. 51-59
[25] Xiang, P., Hou, R., and Zhou, Z., "Cache and Consistency in NOSQL," 3rd IEEE International Conference on Computer Science and Information Technology ICCSIT, 2010. Vol. 6, pp. 117-120
[26] David J. DeWitt and Michael Stonebraker. "MapReduce: A major step backwards." http://databasecolumn.vertica.com/database-innovation/mapreduce-a-major-step-backwards/, diakses pada 10 February 2011.
[27] Mark C. Chu-Carroll, Databases are hammers; MapReduce is a screwdriver, http://scienceblogs.com/goodmath/2008/01/database_are_hammers_mapreduc.php, diakses pada 10 February 2011.
[28] Relational Database Experts Jump The MapReduce Shark, http://typicalprogrammer.com/programming/mapreduce/, diakses pada 10 Feb 2011.