Implementasi ETL dan Perbandingan Performa Column-Oriented Database dan Relational Database sebagai Data Warehouse

Aaron Dwi Caesar Oktaviano, Silvia Rostianingsih, Henry Novianus Palit
Program Studi Teknik Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121 – 131 Surabaya 60236
Telp. (031) – 2983455, Fax. (031) – 8417658
E-Mail: aarondwico@gmail.com, silvia@petra.ac.id, hnpalit@petra.ac.id

ABSTRAK

Implementasi *Non-Relational Database* sebagai *Data Warehouse* dengan menggunakan *Relational Data Warehouse* sebagai sumber menjadi dasar dalam pengerjaan skripsi ini. *Non-Relational* dan *Relational Database* memiliki asumsi yang berbeda dari segi model datanya, sehingga transformasi dari *Relational Data Warehouse* tidak akan membawa hasil yang maksimal. Dalam pengerjaan implementasi ini, *Non-Relational Database* yang akan digunakan bertipe *Column-Oriented/Wide-Column Database*, yaitu *Cassandra*. Perbandingan dengan Relational Data Warehouse akan dilakukan dari segi *latency query, memory usage,* dan *cpu usage*.

Implementasi ini akan dilakukan dengan membuat terlebih dahulu model data yang sesuai dengan tiap database yang digunakan, serta dilakukan ETL untuk memasukkan data. Kemudian, akan diujikan 7 query atau permasalahan untuk menjawab kebutuhan reporting perspustakaan Universitas Kristen Petra.

Berdasarkan hasil pengujian, didapat beberapa kesimpulan. Pemodelan data untuk Column-Oriented Database membutuhkan perhatian lebih, karena kebutuhan pengubahan data dapat mempengaruhi desain dari tabel. ETL juga perlu perhatian, karena dengan tidak adanya fungsi commit, maka perlu diulang dari awal jika gagal. Column-Oriented Database menunjukkan hasil yang lebih baik pada query dengan kondisi yang banyak, karena data bisa semakin cepat ditemukan dan dikembalikan.

Kata Kunci: *Data Warehouse, NoSQL, Cassandra, Column-Oriented Database, Query processing*

ABSTRACT

Implementation of Non-relational Database as Data Warehouse, but using Relational Data Warehouse as source is the basic idea of this thesis. Non-relational model have different assumption than Relational model regarding its data model, using Relational model as source will not yield maximum result. In this implementation, Non-relational Database that will be used is a Column-Oriented/Wide-Column Database, specifically Cassandra. Comparison with Relational Data Warehouse will cover query's latency, memory usage, and cpu usage.

This implementation will begin with making all appropriate data model for each database, and ETL is done to load data into each database. Then, 7 query that is answering Petra Christian University library's need of reporting will be used as testing query.

Some conclusion can be drawn from this test. Data Modeling of Column-Oriented Database need extra cautions, as the need of changing data will affect its design. The ETL process also need extra cautious, as no commit function is present, the error or crash during the process will need the ETL to be restarted from the beginning. Column-Oriented Database showed better result with a lot of where clause, as the data can be found and returned faster

Keywords: *Data Warehouse, NoSQL, Cassandra, Column-Oriented Database, Query processing*

1. PENDAHULUAN

Penggunaan Non-relational Database sebagai Data Warehouse belum memiliki standar sebagaimana Relational Database sebagai Data Warehouse. Penelitian sebelumnya [1] telah menunjukkan data modeling untuk reporting dengan menggunakan Document Database, terutama MongoDB sebagai backend dari Data Warehouse. Penelitian lain, oleh [8], telah melakukan implementasi Data Warehouse dengan Column Oriented Database, yaitu Apache Cassandra, perbandingannya terhadap Document Database sebagai Data Warehouse, serta telah menetapkan pattern dalam melakukan pemodelan Data Warehouse dari Relational Database ke Apache Cassandra.

Dari beberapa penelitian yang ditemukan, belum ada yang melakukan pemodelan ETL (Extract-Transform-Load) secara langsung dari OLTP Relational Database menuju ke Column Oriented Database. Pemodelan ini diperlukan karena berdasarkan teknik yang diajukan oleh [8], proses transformasi memiliki kekurangan dimana diperlukan 2 kali proses, yaitu proses ETL dari OLTP Database ke Relational Data Warehouse, serta proses transformasi Schema dari Relational Data Warehouse menuju ke Column-Oriented Database. Kedua proses ini dilakukan dengan asumsi penggunaan model Relational Database pada NoSQL Database, sehingga model secara physical dari Column-Oriented Database tidak dipergunakan.

2. DASAR TEORI

2.1 Data Warehouse

Data Warehouse merupakan suatu Database yang di-desain untuk keperluan query dan analisa data, bukan untuk memproses transaksi pada umumnya [4]. Data Warehouse memiliki 4 ciri utama, yaitu Subject-Oriented (berfokus pada subjek, misalnya customer), Integrated (mampu menggabungkan data dari berbagai sumber), Non-Volatile (Tidak ada perubahan untuk data yang sudah masuk), dan Time-Variant (Hasil dari Data Warehouse memfokuskan hasil query terhadap suatu waktu).

2.2 Non-relational Database

Non-relational Database merujuk pada penggunaan database yang tidak memodelkan data ke dalam bentuk tabel. NoSQL memiliki keunggulan dari Relational Database pada umumnya terutama dalam hal kemudahan deployment, tingkat read/write yang tinggi, serta kemudahannya dalam hal scaling, dengan

menambahkan node. Ada 4 macam Non-relational Database (NoSQL), yaitu Document Database, Column Oriented Database, Key-Value Database, dan Graph Database. Document Database (MongoDB, CouchDB) memodelkan data sebagai suatu kesatuan dokumen, umumnya dalam bentuk JSON . Column-Oriented Database (Apache HBase, Apache Cassandra) memodelkan dokumen ke dalam bentuk kolom yang diletakkan dalam blok-blok storage . Key-Value Database (Riak, Redis) menyimpan data dalam sebuah Tabel Hash raksasa. Graph Database (Neo4J, InfoGrid) memodelkan data dan relasinya sebagai kumpulan nodes dan edges[7]. Beberapa contoh model dapat dilihat pada Gambar 1 [6]

Gambar 1. Beberapa model NoSQL database [6]

2.3 NoSQL Database Sebagai Alternatif Relational Data Warehouse

Penelitian oleh [1] menjelaskan penggunaan NoSQL, lebih spesifik yaitu Document Oriented Database, sebagai Data Warehouse. Berdasarkan penelitian yang dilakukan, Bicevska et al mengemukakan bahwa permasalahan dalam implementasi Data Warehouse dengan Document Database adalah relasi M:N dalam Relational Database. Dalam Document Database, relasi 1:N pada Document dilakukan dengan memasukkan/embed Document ke Document yang lain, atau dengan reference Document. Relasi M:N tidak bisa dimodelkan ke 2 model tersebut, sehingga model data perlu didenormalisasikan.

Penelitian oleh [2] menjelaskan bahwa Column-Oriented / Wide-Column NoSQL memberikan suatu model yang paling cocok untuk suatu data warehouse dibandingkan dengan model NoSQL lainnya. Diajukan 3 model Schema dalam menggunakan Column-Oriented NoSQL sebagai Data Warehouse. Model pertama disebut Normalized Logical Approach (NLA). Model ini sama seperti model Schema pada Relational Data Warehouse (Fact dan Dimension disimpan terpisah) pada umumnya, namun diimplementasikan pada Column-Oriented NoSQL. Namun karena tidak adanya Referential Integrity pada Column-Oriented NoSQL, maka bagian ini harus ditangani di level aplikasi. Model kedua disebut Denormalized Logical Approach (DLA). Di model ini, semua tabel Fact dan Dimensi dijadikan suatu Big Fact Table (BFT), semua nilai dari Fact dan Dimension dijelaskan oleh suatu identifier (row key), sehingga join tidak diperlukan lagi. Model ketiga disebut Denormalized Logical Approach - Column Family (DLA-CF). Model ini mirip dengan DLA biasa, namun setiap Fact dan Dimension pada BFT ini disimpan sebagai Composite Attribute, setiap Dimension akan dijadikan suatu Column-Family sendiri sehingga oleh Column-Oriented Database dapat disimpan bersamaan, sehingga mempercepat pembacaan data.

Berdasarkan [3] dan [5], penentuan penggunaan Primary Key saat berimbas pada performa query yang dilakukan. Karena pada Column-Oriented Database, primary key akan membantu Engine dari database itu sendiri untuk menentukan lokasi data yang diperlukan, serta untuk pengurutan data secara physical oleh Database. Hal ini memiliki 2 kegunaan yaitu dapat membaca data kebutuhan query jika data terletak dalam partisi yang berbeda, dan penyimpanan dengan *sorting* dapat membuat pembacaan data secara urut, bukan random, yang akan mempercepat query.

3. DESAIN SISTEM

3.1 Analisis Perbedaan dengan Relational Data Warehouse

Pembuatan Schema pada *Relational Database* memiliki standar yang cukup jelas, biasanya disebut *Star Schema*. *Star Schema* memiliki desain menyerupai bintang, dengan *Facts* diletakkan di tengah dan *Dimension* mengelilingi Facts yang ada.

Pembuatan Schema pada Column-Oriented/Wide-Column Data Warehouse akan menyesuaikan query yang dibutuhkan dengan model secara physical dari Database ini. *Primary key* dalam *Column-Oriented* memiliki 2 bagian, yaitu *Partition Key* (letak data pada cluster Database, untuk mencari data) dan *Clustering Key* (model sorting data secara physical, mirip bentuk kolom). *Partition key* menggunakan hashing function dalam peletakannya, sehingga *Partition Key* cocok untuk atribut dengan predicate "=", misalnya jurusan anggota dan kategori buku. Sedangkan *Clustering Key* karena melakukan sorting data, cocok untuk atribut predicate non-equal (">","">="","<="","<")

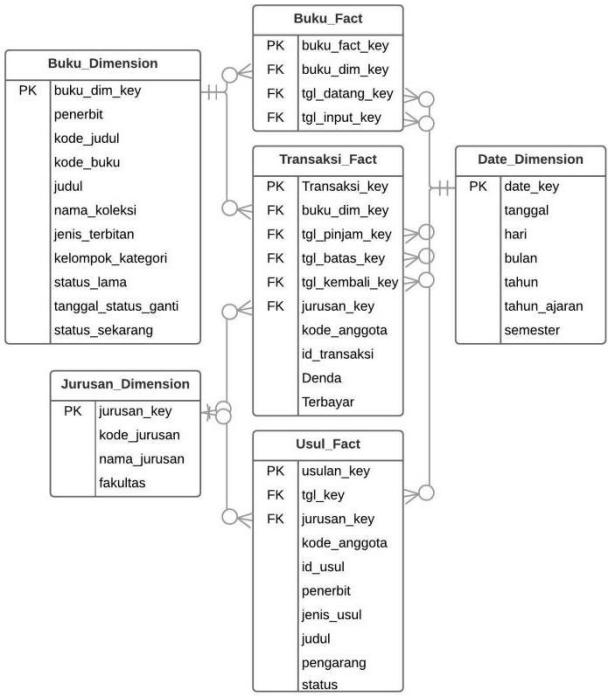
Solusi yang ditawarkan dan dijelaskan di bawah ini adalah pembuatan beberapa tabel sesuai query yang dibutuhkan, dimana setiap tabel akan menyimpan seluruh data yang dibutuhkan seperti dalam bentuk DLA dan DLA-CF. Pembagian tabel didasarkan pada penujuan query yang membutuhkan filter/range yang kurang lebih sama akan ditujukan ke suatu tabel. Penentuan partition key akan menggunakan salah satu filter target dan granularity waktu yang sering digunakan dalam pelaporan.

ID awal dari OLTP database tidak dijadikan *Clustering Key/Unique Value* dikarenakan nilai pada kolom ini tidak bisa di-search secara langsung, meskipun jika memakai *allow filtering,* sehingga dalam pengubahan data nanti akan membutuhkan load data secara keseluruhan. Hal ini juga berlaku untuk implementasi SCD, yaitu kondisi buku, karena dibutuhkan untuk predikat dari query nantinya, sehingga didesain menggunakan *Materialized View.* Untuk *unique-key/surrogate key.* diberikan nilai unique_id (tipe uuid) sebagai *Clustering Key* dengan level terendah.

3.2 Schema Data Warehouse untuk Relational dan Column-Oriented

Untuk *Relational Data Warehouse*, dibuat *star schema* dengan 3 *dimension table,* yaitu jurusan_dim, date_dim, dan buku_dim, serta 3 *fact table*, yaitu buku_fact, transaksi_fact, dan usul_fact. Skema ini dapat dilihat pada Gambar 3

Sedangkan untuk *Column-Oriented Data Warehouse*, dikarenakan asumsi desain dari Column-Oriented Database untuk tidak melakukan join, sehingga semua data harus dijadikan 1 tabel (DLA[]) sesuai kebutuhan query (jika memungkinkan). Dibuat 5 tabel, yaitu cf_transaksi_standard, cf_buku, cf_transaksi_jurusan_kategori, mv_transaksi_kondisi_koleksi, dan cf_usulan. Untuk mv_transaksi_kondisi_koleksi merupakan *materialized view* dari cf_transaksi_jurusan_kategori, karena data yang dipakai sama, hanya berbeda cara akses. Dapat dilihat pada Gambar 4



Gambar 3. Schema Relational Data Warehouse

4. IMPLEMENTASI SISTEM

Implementasi sistem dilakukan pada 3 komputer dengan spesifikasi:

- RAM: 16GB, DDR3
- HDD: 1 TB
- CPU: Core i5
- OS Ubuntu Server 16.04 LTS

Untuk *Relational Database*, akan menggunakan *PostgreSQL single node* dan *PostgresXL 3 node*. Penggunaan 2 macam *Relational Database* ini digunakan untuk melihat bagaimana nantinya Column-Oriented Database dibandingkan dengan *Relational Database* yang biasa dengan yang bertipe *MPP* (Massive Parallel Processing) yang memang dibuat untuk keperluan Data Warehouse ini. Sedangkan untuk *Column-Oriented Database*, *Cassandra*, akan menggunakan 3 node. Namun karena tidak memiliki secara *native support* untuk *join*, maka akan menggunakan *Apache Spark*, namun dengan tidak menyalakan caching untuk mellihat bagaimana *Cassandra* bisa menyediakan data yang dibutuhkan.

ETL dilakukan dengan software *Talend Open Studio*. ETL pada Relational Data Warehouse memiliki keunggulan dimana transformasi dapat dilakukan pada pembuatan dimension dan fact secara terpisah, sehingga ETL menjadi lebih sederhana. Serta adanya commit memastikan data tidak terkirim sebagian saja (atomik). ETL pada Column-Oriented Database harus melakukan setiap transformasi pada setiap ETL yang ada, meskipun data yang ditransformasi dan hasil transformasi sama untuk setiap ETL, serta tidak adanya *commit* menyebabkan adanya error/crash dapat menyebabkan data warehouse tidak dalam kondisi konsisten.

5. ANALISA DAN PENGUJIAN

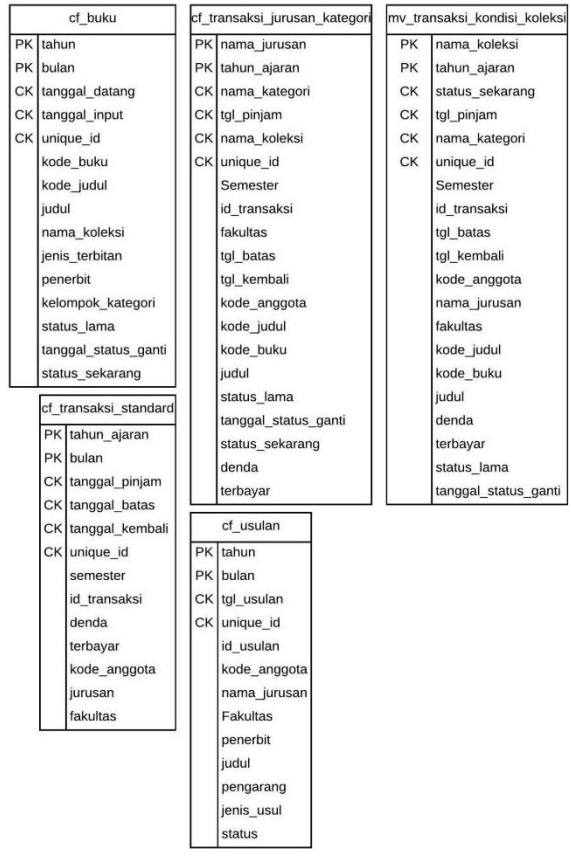
5.1 Query Pengujian

Perpustakaan Universitas Kristen Petra memiliki kebutuhan untuk mendapatkan beberapa hal, misalnya jumlah peminjaman (untuk akreditasi), jumlah denda (untuk laporan keuangan), detail peminjam (buku/kategori/jurusan untuk kebutuhan promosi ke mahasiswa baru), kondisi buku (untuk menambah/mengganti buku yang sering dipinjam/sudah rusak),dsb. berdasarkan kebutuhan ini, dibuat 7 query yang akan diuji di masing-masing sistem, sebagai berikut:

1. Tiga mahasiswa/dosen yang menjadi peminjam terbanyak per tahun ajaran pada tahun ajaran 2014/2015 - 2016/2017 beserta peringkatnya
2. Total denda terkumpul dan yang belum terbayar pada tahun ajaran 2015/2016 per bulannya
3. Total buku yang dimiliki pada tahun 2015 sampai tahun 2016, beserta kenaikan tahunannya pada bulan yang sama tiap tahunnya
4. 5 judul buku dengan peminjaman terbanyak oleh jurusan teknik arsitektur dengan kategori “Geologi”, “Tata kota dan pertamanan”, dan jumlah pinjamannya, beserta jenis koleksi dan statusnya pada tahun ajaran 2013/2014 dan 2015/2016
5. Jumlah pinjaman buku oleh fakultas teknologi industri per semester dan selisih dengan rata-rata tahunannya pada tahun ajaran 2013/2014 hingga 2015/2016
6. 10 buku dengan jenis koleksi‘Referensi’ atau ‘Laporan Kerja Praktek’ yang paling sering dipinjam dan kondisinya sudah hilang pada tahun ajaran 2014/2015 - 2015/2016
7. Penerbit yang paling banyak diusulkan namun usulan masih dalam status “Buku/AV sedang diolah” pada tahun 2013 dan seterusnya

5.2 Penentuan Parameter pengujian

Penentuan parameter akan dilakukan pada duplikasi data 200x dan 400x pada setiap database. Testing parameter PostgreSQL



Gambar 4. Schema Column-Oriented Data Warehouse

dapat dilihat pada Tabel 1 dan Tabel 2. Testing parameter PostgresXL dapat dilihat pada Tabel 3 dan Tabel 4. Testing parameter Cassandra dapat dilihat pada Tabel 5 dan Tabel 6. *memory dan cpu usage PostgreSQL* dan *PostgresXL* dapat dilihat pada Gambar 7. *memory* dan *cpu usage Cassandra* dapat dilihat pada Gambar 8.

Tabel 1. Testing paramater PostgreSQL pada duplikasi 200x

200x Duplikasi					
shared_buffers	**work_mem**	**Test 1**	**Test 2**	**Test 3**	**Average**
128 MB	4 MB	17.173671	17.150233	17.116058	17.15
	16 MB	17.13	17.127042	17.126158	**17.13**
	64 MB	17.113193	17.139266	17.130354	**17.13**
1 GB	4 MB	17.4251	17.375465	17.419326	17.41
	16 MB	17.410567	17.383239	17.426661	17.41
	64 MB	17.440749	17.434036	17.453516	17.44
4 GB	4 MB	17.386109	17.371049	17.364171	17.37
	16 MB	17.388704	17.378163	17.339487	17.37
	64 MB	17.385954	17.379376	17.328916	17.36

Tabel 2. Testing paramater PostgreSQL pada duplikasi 400x

400x Duplikasi					
shared_buffers	**work_mem**	**Test 1**	**Test 2**	**Test 3**	**Average**
128 MB	4 MB	15.5009	15.491853	15.48176	15.49
	16 MB	15.560825	15.572847	15.498528	15.54
	64 MB	15.5032	15.474483	15.494573	15.49
1 GB	4 MB	16.985138	15.436849	15.382363	15.93
	16 MB	15.414987	15.39168	15.39884	**15.4**
	64 MB	15.438906	15.413098	15.4311	15.43
4 GB	4 MB	15.516393	15.504709	15.503966	15.51
	16 MB	15.520499	15.505917	15.503132	15.51
	64 MB	15.454299	15.497597	15.482876	15.48

Tabel 3. Testing paramater PostgresXL pada duplikasi 200x

200x Duplikasi					
shared_buffers	**work_mem**	**Test 1**	**Test 2**	**Test 3**	**Average**
128 MB	4 MB	5.7001	5.672958	5.711465	5.69
	16 MB	5.615787	5.591153	5.601993	5.6
	64 MB	5.605469	5.594746	5.553865	**5.58**
1 GB	4 MB	5.692879	5.658862	5.657863	5.67
	16 MB	5.560634	5.60198	5.589888	**5.58**
	64 MB	5.557643	5.590827	5.600512	**5.58**
4 GB	4 MB	5.687713	5.702641	5.700163	5.7
	16 MB	5.678513	5.687964	5.721292	5.7
	64 MB	5.680804	5.726087	5.656063	5.69

Tabel 4. Testing paramater PostgresXL pada duplikasi 400x

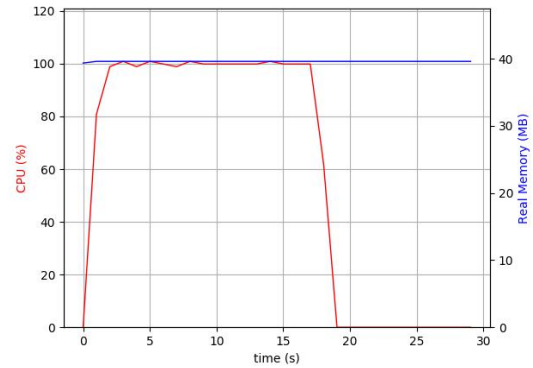
400x Duplikasi					
shared_buffers	**work_mem**	**Test 1**	**Test 2**	**Test 3**	**Average**
128 MB	4 MB	11.101507	11.127557	11.126492	**11.12**
	16 MB	11.398783	11.418275	11.410011	11.41
	64 MB	11.339427	11.305663	11.340745	11.33
1 GB	4 MB	11.130715	11.093385	11.136219	**11.12**
	16 MB	11.215624	11.168472	11.209792	11.2
	64 MB	11.271551	11.255313	11.265251	11.26
4 GB	4 MB	11.286754	11.284454	11.260903	11.28
	16 MB	11.391281	11.382052	11.350582	11.37
	64 MB	11.393421	11.404565	11.38595	11.39

Tabel 5. Testing paramater Cassandra pada duplikasi 200x

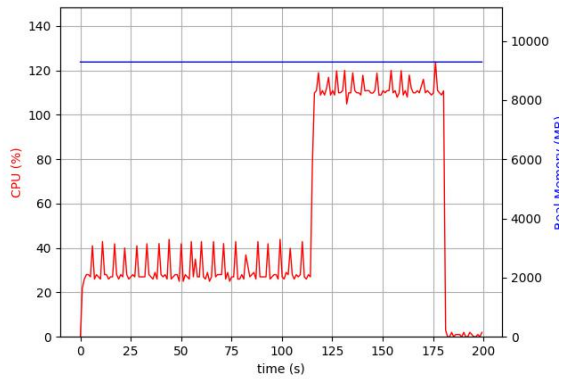
executor number	executor core	executor memory	MAX HEAP SIZE	Test 1	Test 2	Test 3	Average
3	4	12	4 GB	10.341798	9.5700033	9.3430631	9.75
3	4	9		9.7523537	9.3346045	9.2479842	**9.44**
3	4	6		9.8923833	9.5154827	9.4226966	9.61
3	4	3		9.8130212	9.4161592	9.14364	9.46
3	4	12	8 GB	10.596324	9.901413	9.7006347	10.07
3	4	9		10.495139	10.138638	9.7548718	10.13
3	4	6		10.319704	9.8722992	9.6955643	9.96
3	4	3		10.040517	9.652137	9.2639136	9.65
6	2	6	4 GB	12.5416	10.262193	9.9010425	10.9
6	2	3		12.941997	10.180291	9.9566624	11.03
6	2	6	8 GB	13.176584	10.409435	9.7332714	11.11
6	2	3		12.483429	10.174498	9.8979287	10.85

Tabel 6. Testing paramater Cassandra pada duplikasi 400x

executor number	executor core	executor memory	MAX HEAP SIZE	Test 1	Test 2	Test 3	Average
3	4	12	4 GB	17.469495	16.63223	16.452682	16.85
3	4	9		17.345017	16.41155	16.623426	16.79
3	4	6		16.780961	16.706043	16.163955	**16.55**
3	4	3		17.401031	16.477462	16.592495	16.82
3	4	12	8 GB	17.46731	17.256667	17.204933	17.31
3	4	9		17.444055	17.413415	16.925707	17.26
3	4	6		17.558818	17.210727	16.704813	17.16
3	4	3		17.371464	16.922217	16.710015	17
6	2	6	4 GB	19.653009	16.904384	16.877349	17.81
6	2	3		17.605369	19.409415	16.928075	17.98
6	2	6	8 GB	20.230578	17.406686	17.236566	18.29
6	2	3		17.670324	20.116785	17.312038	18.37



Gambar 7. *memory usage, cpu usage,* dan *latency* pada PostgreSQL dan PostgresXL saat penentuan parameter



Gambar 8. *memory usage, cpu usage,* dan *latency* pada Cassandra saat testing parameter

Untuk *PostgreSQL*, terlihat dengan semakin banyaknya duplikasi, maka latency menjadi lebih kecil dengan *shared_buffers* yang cukup besar, sedangkan untuk *work_mem* sudah mencukupi, dan diambillah konfigurasi *shared_buffers* 1GB dan *work_mem* 16MB. Di sini PostgreSQL terlihat memiliki keunggulan dimana query planner dapat melihat statistik data dan menentukan hasil yang lebih cepat. Untuk *PostgresXL*, karena terpecah di 3 node, pada pencobaan 200x belum terlihat konfigurasi yang baik dengan hasil yang cukup sama di tiap konfigurasi. Sedangkan pada 400x, terlihat bahwa hanya 2 nilai *shared_buffers*, yaitu 128MB dan 1GB pada *work_mem* 4MB. Mengingat testing akan dilakukan pada duplikasi 1000x, maka *shared_buffers* 1GB dan *work_mem* 4MB yang akan dipakai. Untuk *Cassandra* dan *Spark*, terlihat bahwa dengan meningkatkan duplikasi, maka diperlukan *memory* lebih banyak pada *executor spark*. Namun jika memory total yang digunakan melebihi kapasitas memory komputer, maka hasil menjadi tidak maksimal. Operating System tetap membutuhkan resource untuk mengontrol program dsb, sehingga penggunaan kombinasi yang terlalu memaksimalkan angka memori tidak menghasilkan performa maksimal. Karena itu, maka akan dipilih konfigurasi *Cassandra* dengan MAX_HEAP_SIZE 4GB, dan konfigurasi *Spark* adalah 3 *executor*, masing-masing 4 *cores* dan *memory* 6GB.

5.3 Pengujian Akhir

Setelah parameter ditentukan, maka testing dilakukan pada duplikasi 1000x. Namun, untuk melihat perubahan yang terjadi, testing juga dilakukan pada data awal / tanpa duplikasi. Testing akhir pada PostgreSQL dapat dilihat pada Tabel 7. Testing akhir pada PostgresXL dapat dilihat pada Tabel 8. Testing akhir pada Cassandra dapat dilihat pada Tabel 9. *memory* dan *cpu usage PostgreSQL* saat testing akhir dapat dilihat pada Gambar 9. *memory* dan *cpu usage PostgresXL* saat testing akhir dapat dilihat pada Gambar 10. *memory* dan *cpu usage Cassandra* saat testing akhir dapat dilihat pada Gambar 11.

Tabel 7. Hasil Testing pada PostgreSQL

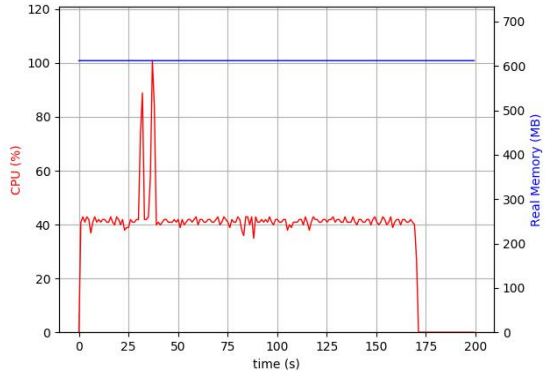
PostgreSQL		Test 1	Test 2	Test 3	Average
Duplikasi 1	Q1	0.172271	0.171928	0.171379	0.17
	Q2	0.097171	0.097464	0.097278	0.1
	Q3	0.116183	0.116459	0.11615	0.12
	Q4	0.169995	0.16984	0.130745	0.16
	Q5	0.081495	0.08053	0.080808	0.08
	Q6	0.340227	0.342039	0.341196	0.34
	Q7	0.005994	0.004709	0.004779	0.01
Duplikasi 1000	Q1	1346.396178	1353.959861	1340.108025	1346.82
	Q2	172.365204	172.143013	172.296306	172.27
	Q3	108.379705	108.488098	108.28175	108.38
	Q4	169.747085	169.877107	169.598885	169.74
	Q5	169.387986	169.337075	169.40464	169.38
	Q6	169.016197	169.288914	168.889461	169.06
	Q7	0.871038	0.945376	0.908016	0.91

Tabel 8. Hasil Testing pada PostgresXL

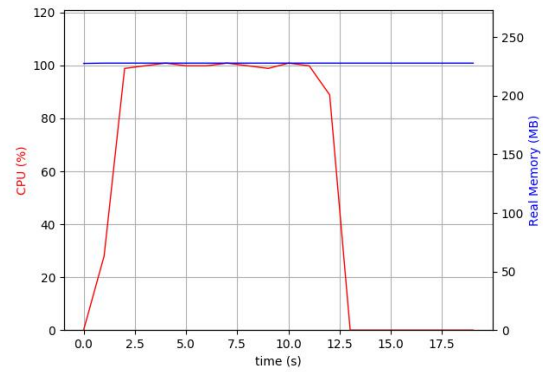
PostgresXL		Test 1	Test 2	Test 3	Average
Duplikasi 1	Q1	0.176338	0.16974	0.173423	0.17
	Q2	0.044179	0.043678	0.044661	0.04
	Q3	0.047125	0.053604	0.059699	0.05
	Q4	0.076528	0.076538	0.075879	0.08
	Q5	0.036354	0.03782	0.037953	0.04
	Q6	0.150523	0.149093	0.152086	0.15
	Q7	0.008237	0.009039	0.008707	0.01
Duplikasi 1000	Q1	372.883655	362.341352	360.328996	365.18
	Q2	18.245	18.253971	18.246153	18.25
	Q3	21.01868	20.748676	20.736753	20.83
	Q4	11.2564	11.24574	11.271655	11.26
	Q5	11.858828	11.854196	11.890612	11.87
	Q6	10.204452	10.19463	10.202127	10.2
	Q7	0.335062	0.335347	0.336897	0.34

Tabel 9. Hasil Testing pada Cassandra

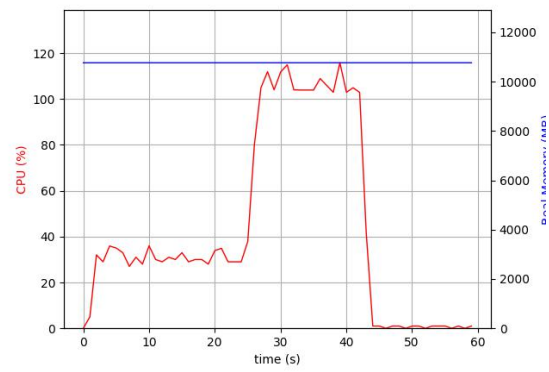
Cassandra		Test 1	Test 2	Test 3	Average
Duplikasi 1	Q1	2.2413	1.8297	1.7136	1.93
	Q2	1.202	1.101	1.1757	1.16
	Q3	1.4979	1.2059	1.1689	1.29
	Q4	1.146	1.1665	0.9771	1.1
	Q5	1.3016	1.1528	1.0847	1.18
	Q6	0.6293	0.6207	0.6116	0.62
	Q7	0.8417	0.7446	0.7717	0.79
Duplikasi 1000	Q1	513.4433837	472.0534742	454.5384226	480.01
	Q2	429.369313	394.5343447	392.6091292	405.5
	Q3	238.4999092	237.0621059	235.6983171	237.09
	Q4	42.07644224	41.21544552	41.01284242	41.43
	Q5	198.5619214	200.3057384	200.3173203	199.73
	Q6	1.027056217	0.964957952	0.97654748	0.99
	Q7	21.14004803	21.38770485	21.69156313	21.41



Gambar 9. *memory usage, cpu usage,* dan *latency* pada testing Cassandra



Gambar 10. *memory usage, cpu usage,* dan *latency* pada testing Cassandra



Gambar 11. *memory usage, cpu usage,* dan *latency* pada testing Cassandra

Dari hasil testing query, dapat dilihat bahwa pada duplikasi 1x, Cassandra dan Spark menunjukkan *latency* query yang paling lama, yaitu sekitar 1-2s. Sedangkan untuk yang lainnya berada

di bawah 1s. Pada level duplikasi ini juga, setiap quey pada suatu platform belum terlihat bedanya dengan query lain dari segi waktu. Untuk *memory usage* dan *cpu usage*, kurang lebih sama dengan ketika testing. Satu-satunya yang berbeda adalah *memory usage* pada PostgreSQL, memory usage mencapai 80 MB, hal ini dikarenakan data yang sudah diduplikasi, dan PostgreSQL akan load data index ke memory lebih dulu. Data yang lebih banyak ini menyebabkan *memory usage* yang lebih besar.

Sedangkan pada duplikasi 1000x, ada beberapa hal yang bisa diperhatikan. Pertama, terlihat pada semua platform kalau query nomor 1 yang menjadi paling lama pada PostgreSQL dan PostgresXL. Hal ini disebabkan query 1 memiliki banyak variabel pada bagian *group by*, yang bisa dioptimize dengan mengeluarkan data jurusan ke query luar. Hal ini tidak memungkinkan pada Cassandra.

Kedua, penggunaan memory pada ketiga database menjadi berbeda. Pada PostgreSQL, *memory usage* menjadi mencapai ~600MB. Pada PostgresXL, setiap node memiliki *memory usage* menjadi ~250MB. Dan pada Cassandra, meskipun MAX_HEAP_SIZE diset 8GB, *memory usage* menjadi pada ~12GB. Hal ini menyebabkan Spark hanya mendapat memori sekitar ~600MB-1.8GB. *memory usage, cpu usage,* dan *latency* PostgreSQL pada duplikasi 1000x dapat dilihat pada Gambar 5.4. *memory usage, cpu usage,* dan *latency* PostgresXL pada duplikasi 1000x dapat dilihat pada Gambar 5.5. *memory usage, cpu usage,* dan *latency* Cassandra dan Spark pada duplikasi 1000x dapat dilihat pada Gambar 5.6 dan Gambar 5.7.

Ketiga, PostgreSQL dan PostgresXL pada query 2, 3, 5, dan 7 lebih cepat daripada Cassandra, dengan PostgresXL dalam ~20s, PostgreSQL dalam ~140-160s dan Cassandra pada ~200-500s. Pada query 1, PostgresXL tetap yang paling cepat, sedangkan PostgreSQL melonjak hingga ~1300-1400s. Cassandra tetap pada ~450-550s. Pada query 4, PostgresXL menjadi paling cepat, yaitu ~10-11s. PostgreSQL berada pada ~169s, dan Cassandra pada ~40-42s. Pada query 6, Cassandra menjadi paling cepat, yaitu ~2s. PostgreSQL berada pada ~169s, dan PostgresXL pada ~10-11s.

Hal menarik yang terlihat adalah latency query pada Cassandra turun secara drastis pada query 4 dan 6, yaitu query yang ditujukan pada tabel yang sesuai. Namun, terlihat juga pada query 5, meskipun menggunakan tabel yang sesuai, namun jika data yang dibaca cukup banyak, maka query akan tetap agak lama. Dapat diartikan bahwa Cassandra sebagai Data Warehouse akan sesuai untuk kasus *drill-down* yang cukup dalam, karena data bisa dicari dengan sangat cepat.

6. KESIMPULAN

Dari hasil pengerjaan Column-Oriented / Wide-Column Database sebagai Data Warehouse, disimpulkan beberapa hasil, yaitu:

- Pemodelan data pada Column-Oriented Database membutuhkan perhatian lebih, karena kebutuhan pengubahan data mempengaruhi dan membutuhkan penyesuaian desain dari tabel tersebut. jika nantinya dibutuhkan perubahan, karena primary key tidak bisa diubah dan tidak bisa dicari secara langsung. Penggantian data (SCD pada relational) memerlukan sangat banyak data untuk diubah dan perlu disesuaikan dengan primary key tabel, atau penggantian data menggunakan SCD tipe 2 dibanding 3, yaitu dengan menambah data baru
- Dalam proses ETL ke database, pada Column-Oriented Database juga lebih susah dalam memastikan kebenarannya jika terjadi kegagalan. Hal ini disebabkan tidak adanya fungsi *commit* pada *Database* ini, menyebabkan error dapat menghasilkan data tidak lengkap (tidak atomik)
- Column-Oriented Database sebagai Data Warehouse, berfungsi dengan baik pada bagian *drill-down* yang cukup dalam, dimana Database dapat mencari dan mengembalikan data dalam waktu yang lebih cepat tanpa mengecek keseluruhan data. Di sisi lain, dikarenakan modelnya yang tidak memakai join, maka ada query yang tidak bisa dioptimasi, seperti dengan mengeluarkan kriteria join agar tidak memakan *memory* terlalu banyak

Untuk saran penelitian lebih lanjut, dapat dilakukan pada Database tipe lain, terutama pada model database yang memiliki kesamaan dengan *Data Warehouse*, misalnya *search-engine* (*elasticsearch / solr*) karena kedua model tersebut berusaha mencari data dari kumpulan data yang besar, dan melakukan fungsi seperti aggregate data (group by), dsb

7. DAFTAR PUSTAKA

[1] Bicevska, Z, Oditis, I. 2016. Towards NoSQL-based Data Warehousing Solutions. Procedia Computer Science. Vol. 104; p. 104-111

[2] Dehdouh, K., Boussaid, O., Bentayeb, F., Kabachi, N. 2015. Using the Column Oriented NoSQL Model for Implementing Big Data Warehouse. Proceedings of the 21st International Conference on Parallel and Distibuted Processing Techniques and Applications, pp. 469-47

[3] Hobbs, T. 2015. Basic Rules of Cassandra Data Modeling. from https://www.datastax.com/dev/blog/basic-rules-of-cassandra-data-modeling

[4] Lane, P. 2013. Oracle Database Data Warehousing Guides. Oracle Inc

[5] McFadin, P. 2016. The Most Important Thing to Know in Cassandra Data Modeling: The Primary Key. Retrieved September 22, 2018. from https://www.datastax.com/dev/blog/the-most-important-thing-to-know-in-cassandra-data-modeling-the-primary-key

[6] Vorries, B. 2014. Lesson 7: Column Oriented Databases (AKA Big Table or WideColumn). Retrieved July 24, 2018 from http://data-magnum.com/lesson-7-column-oriented-databases-aka-big-table-or-wide-column

[7] Woodey A. 2014. Forrester Ranks the NoSQL Database Vendors. Retrieved October 15, 2018. http://www.datanami.com/2014/10/03/forrrester-ranks-nosql-database-vendors

[8] Yangui, R, Nabli, A, Gargouri, F. 2016. Automatic Transformation of Data Warehouse Schema to NoSQL Data Base: Comparative Study. Procedia Computer Science. Vol. 9 5; p. 255-264 (2016)