Analisis Performansi Metode Graph Decomposition Index pada Graph Database

Isjhar Kautsar
Fakultas Informatika
Telkom University
Bandung, Indonesia
isjhar@gmail.com

Kemas Rahmat Saleh W., S.T., M.Eng
Fakultas Informatika
Telkom University
Bandung, Indonesia
bagindok3m45@gmail.com

Gia Septiana Wulandari, S.Si., M.Sc.
Fakultas Informatika
Telkom University
Bandung, Indonesia
giaseptiana@telkomuniversity.ac.id

***Abstraksi*— Kekurangan relational database yang ditemui, seperti sulitnya membuat desain relational database yang pas, kurang mampu mengakomodir data semi terstruktur, dan kurang mampunya mengakomodir data yang memiliki banyak relasi mendorong para peneliti untuk menemukan model database yang baru. Salah satunya graph database.**

Graph database mampu menjadi solusi berbagai permasalah tersebut. Namun, graph database sendiri masih memiliki beberapa kekurangan, yaitu pencarian dilakukan secara sekuensial pada saat proses retrieve data. Oleh karena itu, suatu metode diperlukan untuk mengatasi masalah ini, yaitu dengan indexing. Metode indexing yang akan digunakan adalah graph decomposition index.

Graph decomposition index cocok dengan jenis data yang digunakan, yaitu pada bagian keunggulannya dalam me-retrieve data query berupa subgraph dan model graph berupa simpel graph berlabel.

Penelitian ini akan membangun dua buah sistem yaitu graph database tanpa index dan graph database dengan index. Kemudian dibandingkan performansinya berdasarkan response time (execution time dan IO time).

Dari penelitian ini diperoleh hasil bahwa graph database dengan index akan memiliki performansi execution time yang lebih bagus dibanding graph database tanpa index, tetapi memiliki IO time yang jelek dibanding graph database tanpa index

Keywords—graph database; indexing; relational; graph decomposition index.

I. PENDAHULUAN

Sulitnya membuat desain database dengan model relational yang pas, meningkatnya jenis data yang semi terstruktur, dan meningkatnya waktu proses query jika jumlah relasi semakin banyak mendorong orang-orang melakukan penelitian untuk menemukan model database yang baru. Salah satu model yang ditemukan dalam penelitian tersebut adalah model graph. Peneliti menemukan bahwa graph database dapat menjadi solusi terhadap permasalahan tersebut [7].

Delapan tahun belakangan ini graph database mulai sering digunakan dalam berbagai bidang industri seperti, kesehatan, media, minyak dan gas, gaming, dllkarena kelebihannya. Menurut prediksi peneliti, graph akanumum digunakan 15 tahun mendatang [7].

Namun, Masalah krusial dan sering terjadi dalam aplikasi yang berbasiskan graph khususnya graph database adalah efisiensi proses retrieve data.Proses retrieve tidak efisien jika dilakukan secara sequential karena proses pencarian tidak hanya dilakukan dengan menelusuri seluruh graph database tetapi juga melakukan proses matching subgraph q dengan subgraph yang ada di graph database [12]. Oleh karena itu, diperlukan suatu metode untuk meningkatkan efisiensi dalam prosesretrieve, yaitu indexing.

Pada tugas akhir ini, metode indexing yang akan digunakan adalah graph decomposition index. Metode ini dipilih karena metode ini memiliki keunggulan dalam me-retrieve data subgraph, sehingga metode ini cocok dengan data yang digunakan yaitu data perpustakaan Telkom University yang jika dimodelkan menjadi graph model, proses retrieve datanya berupa subgraph. Selain itu data perpustakaan Telkom University juga memenuhi batasan dari metode ini[1].

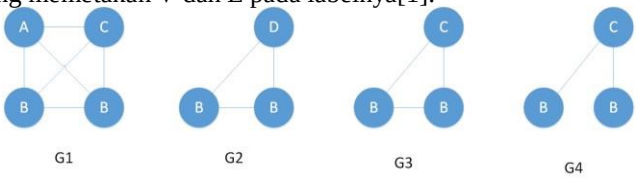
II. LANDASAN TEORI

Berikut konsep-konsep yang digunakan dalam penelitian ini.

A. *Teori Graph*

Secara definisi, sebuah graph terdiri dari kumpulan dari vertex (V) dan edge (E). Pada sebuah graph, sebuah edge minimal berasosiasi dengan satu atau dua buah vertex yang disebut endpoint. Sebuah graph dapat disimbolkan G = (V, E). Sebuah graph yang tidak terdapat dua buah vertex terhubung dengan lebih satu edge dan tidak terdapat edge loop (edge yang menghubungkan ke vertex yang sama) disebut simple graph. Graph yang terdapat dua vertex yang dihubungkan oleh lebih dari satu edge disebut multigraph. Berdasarkan keberadaan arah pada edge pada sebuah graph terdiri dari direct graph dan indirect graph. Direct graph memiliki sebuah vertex yang menjadi start dan sebuah vertex yang menjadi end, sedangkan undirect graph tidak memiliki vertex yang menjadi start dan end[9].

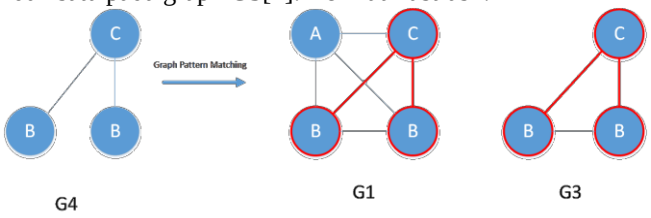
Berdasarkan ada labelnya atau tidak pada vertex dan edge, graph dibedakan atas graph berlabel dan graph tak berlabel. Graph berlabel adalah tuple yang memiliki 4 elemen G = (V, E, Σ, λ) dimana V adalah sekumpulan Vertex, E adalah sekumpulan Edge yang menghubungkan vertex di V, Σ adalah kumpulan label pada vertex dan edge, dan λ adalah fungsi yang memetakan V dan E pada labelnya[1].



Gambar 1. Kumpulan Simple undirect graph berlabel

Simple graph dikatakan isomorphic dengan graph lain jika seluruh vertex dan edge pada graph tersebut berkorespondesi satu-satu dengan seluruh vertex dan edge graph lain.

Pada graph dikenal juga istilah graph pattern matching. Graph pattern matching adalah proses mencari satu atau lebih pola dari sebuah graph yang ada pada graph lain. Misal pada gambar 2-1, G4 adalah pola graph yang akan dicari sedangkan G1, G2, dan G3 adalah graph yang dijadikan target pencarian untuk graph pattern matching, sehingga ditemukan 2 buah graph hasil dari graph pattern matching, yaitu satu pada graph G1 dan satu pada graph G3[2]. Berikut ilustrasi :



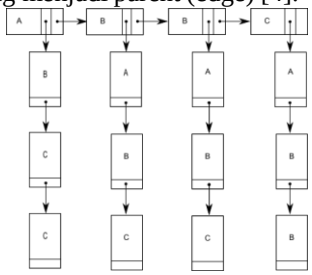
Gambar 2. Graph Pattern Matching G4 terhadap G1, G2, dan G3, yang ada pada gambar 2-1

Ada dua cara untuk merepresentasikan graph, yaitu dengan matriks ketetanggan dan multi linked list. Pada graph dengan jumlah vertex sebanyak n akan membentuk matriks ketetanggan (M) dengan ukuran $n \times n$ dan nilai setiap entry merepresentasikan ada tidaknya keterhubungan antar vertex. Jika ada, maka entry bernilai 1 dan 0 jika tidak ada [4].

	A	B	B	C
A	0	1	1	1
B	1	0	1	1
B	1	1	0	1
C	1	1	1	0

Gambar 3. Representasi matriks ketetangaan graph G1

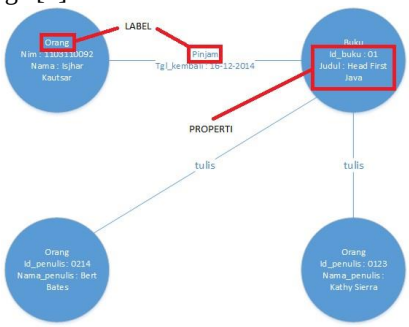
Pada representasi multi linked list, parent list direpresentasikan sebagai vertex yang ada pada suatu graph dan child list direpresentasikan sebagai list vertex yang terhubung dengan vertex yang menjadi parent (edge) [4].



Gambar 4. Representasi multi linked list pada graph G1

B. *Graph Database*

Graph database adalah sebuah database management system yang memiliki kemampuan Create, Read, Update, dan Delete (CRUD) pada data yang dibentuk menggunakan model graph. Pada RDBMS setiap data direpresentasi sebagai sebuah tuple pada tabel, sedangkan pada graph database setiap data direpresentasikan sebagai sebuah vertex. Jika terdapat relasi pada sebuah tuple maka relasi tersebut dibuat dalam bentuk edge yang menghubungkan antar vertex. Pada graph database, terdapat istilah label dan properti. Label adalah kategori dari vertex atau edge dan properti adalah atribut yang ada pada vertex atau edge [6].



Gambar 5. Contoh Graph Database

C. *Indexing*

Indexing adalah struktur yang terorganisasi yang dibuat oleh seorang database administrator yang struktur tersebut berasosisasi dengan data untuk meningkatkan kecepatan akses terhadap data yang ingin dicari. Dengan indexing, data dikelompokkan menjadi beberapa bagian. Pengelompokan biasanya berdasarkan kemiripan setiap data, sehinga proses pencarian tidak dilakukan dengan menelusuri seluruh data, melainkan hanya dengan menelusuri bagian yang mirip dengan yang ingin dicari[13].

File index pada database bekerja seperti index yang ada pada buku, misal textbook. Kita dapat mencari suatu topik dengan melihat index yang ada pada bagian belakang pada buku. Menemukan halaman mana saja kata tersebut bisa ditemukan pada buku, dan mendapatkan informasi apa yang akan kita cari. Dengan index, usaha yang diperlukan dalam proses pencari dapat diminimalkan. Layaknya index pada sebuah buku. Setiap kata yang ada pada buku dikelompokkan berdasarkan huruf pertamanya.

Terdapat dua metode dasar pada index :

1. Ordered Indices : berdasarkan nilai yang telah diurutkan
2. Hash Indices : berdasarkan distribusi uniform dari nilai rentang bucket yang disediakan. Bucket yang digunakan untuk menyimpan data ditentukan menggunakan function, yang disebut hash function.

Di antara kedua metode di atas, tidak ada metode yang paling bagus, kedua-duanya memiliki kelebihan dan kekurangan masing-masing, sehingga penggunaannya tergantung jenis aplikasinya.

D. *Hash Table*

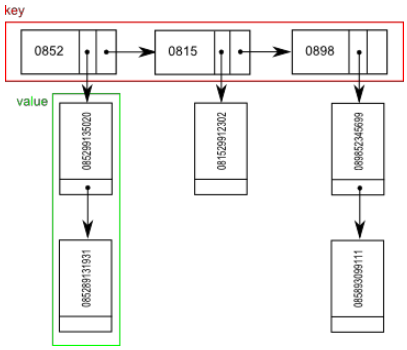
Hash table adalah model struktur data yang digunakan untuk menyimpan informasi dan mempermudah proses retrieve. Hash table terdiri atas dua komponen, yaitu key dan value. Key digunakan untuk mengakses data yang disimpan. Key bersifat unik dan biasanya berupa string yang penentuan nilai string tergantung dengan kasusnya. Hash table biasanya direpresentasikan sebagai associative array. Associative array mirip dengan array, tetapi untuk mengakses nomor index-nya array tidak dapat dilakukan secara langsung menggunakan integer, tetapi dengan menggunakan string dengan bantuan hash function yang gunanya untuk mengubah key yang berupa

string tadi menjadi nilai integer yang menyatakan index array tempat data disimpan. Implementasi associative array sebagai multi linked list, disebut juga chain. Pada pembuatan hash table dengan chain, setiap key akan di-generate menggunakan hash function untuk mendapatkan nilai hash value-nya yang digunakan untuk mendapatkan posisi index data yang ingin diakses. Setiap key akan merujuk hanya ke sebuah entry. Entry berupa linked list yang menyimpan data yang memiliki hasil generati hash value yang sama. Dengan hash table, proses pencarian tidak perlu menelusuri seluruh nilai, cukup menelusuri slot yang memiliki key sesuai dengan value yang ingin dicari [6].

Pada tugas akhir ini, pembangunan sistem menggunakan bahasa pemrograman java, sehingga pembuatan hash table menggunakan class HashMap pada java. Untuk proses penyimpanan dan retrieve data dari HashMap menggunakan method get dan set. Pada saat pemanggilan method get untuk me-retrieve data diperlukan key berupa string sebagai parameter dan pemanggilan method put untuk menyimpan data diperlukan key berupa String dan value berupa Object sebagai parameter. Cara kerja untuk proses get dan put HashMap pada java bekerja sebagai berikut :

untuk method get dan put, key berupa string yang diubah menjadi sebuah nilai integer atau yang disebut hash value yang menyatakan index array tempat penyimpanan data yang disebut bucket dengan menggunakan hash function. Bucket berupa linked list yang menyimpan value. Jadi dua key yang berbeda bisa menghasilkan hash value yang sama ketika dilakukan proses generate menggunakan hash function, sehingga untuk membedakan value data yang satu dengan data yang lain disimpanlah informasi key dari data tersebut.

pencarian data yang ingin dicari dilakukan secara sequential pada bucket dengan cara mencocokkan key parameter dengan key yang ada pada data yang dimiliki. Jika ditemukan data yang memiliki key yang sama dengan key parameter, maka untuk method get data tersebut di return sebagai jawaban sedangkan untuk method put, value pada data tersebut akan di-replace dengan data baru yang akan disimpan, dan jika tidak ditemukan maka untuk method get akan me-return nilai null, sedangkan untuk method put artinya data tidak ditemukan sehingga data baru akan ditambahkan pada akhir list[5].



Gambar 6. Hash table menggunakan multi linked list

E. Graph Canonical Form

Graph canonical form adalah metode untuk merepresentasikan sebuah graph ke dalam bentuk canonical form yang disimbolkan dengan code(M) misal canonical code untuk graph M. Caranya, pertama, sebuah graph dengan jumlah vertex sebanyak n direpresentasikan ke dalam matriks ketetanggan yang dimisalkan dengan matriks M dengan ukuran $n \times n$. Matriks ketetanggaan yang dibuat agak berbeda dengan matrik ketetanggaan pada umumnya. Bedanya, setiap diagonal entry diisi oleh vertex dan entry selain itu diisi oleh edge yang nilainya 1 atau nilai label edge jika ada edge dan 0 jika tidak ada. Setelah itu, tulis isi value dari tiap entry dengan urutan M1,1, M2,1, M3,1, ...Mn,1, M1,2, M2,2, M2,3,..., Mn,n. Pada Mi,j, i direpresentasikan sebagai nomor kolom dan j direpresentasikan sebagai nomor baris. Untuk graph tidak berarah, penulisan value dari tiap entry cukup berfokus pada entry tringular yang dibagian bawah. Selanjutnya, cari seluruh pola code yang dapat dibentuk dari graph tersebut. Kemudian pilih code yang paling maksimal dari seluruh kemungkinan code yang ada. Nilai maksimal dilihat dari nilai terbesar jika dibandingkan secara lexicograhically [3]. Lexicographically adalah metode mengurutkan posisi kata berdasarkan urutan pada alfabet setiap komponen dari sebuah kata [10]. Contoh graph canonical from dari graph G1 pada gambar 2-1. Misal pola yang dapat dibentuk dan kemungkinan bernilai maksimal ada sebanyak 6. Misalkan code ke-1 sampai 6 direpresentasikan dengan code(M1), code(M2), ..., code(M6).

Matriks M_1

C			
1	B		
1	1	B	
1	1	1	A

Matriks M_2

C			
1	A		
1	1	B	
1	1	1	B

Matriks M_3

C			
1	B		
1	1	A	
1	1	1	B

Matriks M_4

B			
1	A		
1	1	B	
1	1	1	C

Matriks M_4

A			
1	B		
1	1	C	
1	1	1	B

Matriks M_5

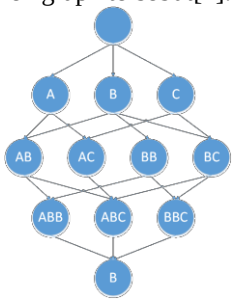
B			
1	C		
1	1	B	
1	1	1	A

Gambar 7. Matrix ketetanggaan (M1, M2, M3, M4, M5, dan M6 untuk graph G1

code(M1) = “C1B11B111A”, code(M2) = “C1A11B111B”, code(M3) = “C1B11A111B”, code(M4) = “B1A11B111C”, code(M5) = “A1B11C111B”, code(M6) = “B1C11B111A”. Berdasarkan lexicographically code C > B > A > 1 > 0, sehinga yang menjadi code dengan nilai maksimum yang merepresentasikan matriks M adalah code(M1)

F. Graph Decomposition

Graph decomposition adalah metode untuk menguraikan/dekomposisi sebuah graph menjadi graph yang lebih kecil untuk menelurusi segala kemungkingkinan subgraph yang dapat dibentuk oleh graph tersebut[1].



Gambar 8. Graph Decomposition dari graph G1

G. Graph Decomposition Index

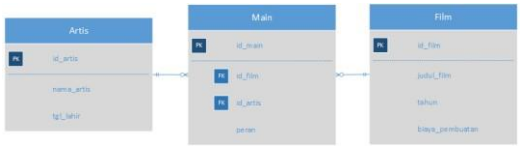
Seluruh subgraph yang didapatkan dari proses graph decomposition di-index menggunakan hash table. Proses hash graph dilakukan dengan merepresentasikan setiap graph menjadi canonical form. Hash key kemudian ditentukan dari canonical code setiap graph. Dengan ini, seluruh graph yang isomorphic akan menghasilkan hash key yang sama. Seluruh entry pada hash tabel berupa graph dan hanya satu entry yang diperuntukan untuk setiap canonical code yang unik. Proses retrieve dilakukan dalam dua tahap. Pertama, query yang telah diubah menjadi bentuk canonical code akan dicari hash key-nya. Proses tersebut akan menghasilkan kandidat-kandidat graph yang kemungkinan cocok dengan canonical code dari query. Selanjutnya proses pencocokan dilakukan terhadap kandidat-kandidat graph yang kemungkinan cocok tadi dengan query. Jika ditemukan kandidat yang cocok dengan query, maka kandidat itu adalah jawaban dari query [1].

H. Transformasi Data Relational Model ke Graph Model

Transformasi data dari *Relational* ke *Graph model* dapat dibagi menjadi dua. Yaitu untuk entitas yang berelasi dengan entitas lain dengan kardinalitas banyak ke banyak dengan banyak ke satu atau satu ke satu[8].

1) Transformasi dengan kardinalitas relasi antar entitas banyak ke banyak

Gambar 9. ERD relasi banyak ke banyak



Gambar 10. Tabel relasi entitas banyak ke banyak

Tahapan transformasi untuk entitas yang berelasi banyak ke banyak proses transformasinya sebagai berikut :

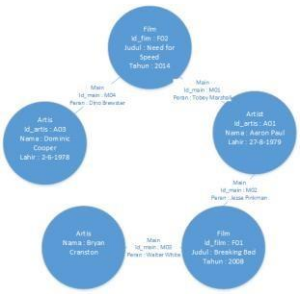
1. ubah semua row pada tabel menjadi vertex dengan label vertex sama dengan nama tabel, kecuali row pada tabel yang menghubungkan dua tabel diikuti nilai setiap atribut pada row menjadi nilai properti pada vertex dengan catatan nilai null tidak perlu dijadikan properti pada vertex. Pada contoh gambar 2-10, semua row pada tabel artis dan film yang diubah menjadi vertex, dan
2. ubah semua row yang ada pada tabel yang menjadi penghubung dua entitas menjadi edge yang menghubungkan kedua vertex entitas tersebut dengan label edge sama dengan nama tabel diikuti nilai atribut yang menjadi properti pada edge dengan catatan nilai null dan foreign key tidak perlu dijadikan properti. Pada contoh gambar 2-10 tabel main menjadi edge yang menghubungkan vertex artis dan film pada graph database.

artis		
Id_artis	Nama_artis	Tgl_lahir
A01	Aaron Paul	27-8-1979
A02	Bryan Cranston	null
A03	Dominic Cooper	2-6-1978

film		
Id_film	Judul_film	Tahun
F01	Need for speed	2014
F02	Breaking Bad	2008

main			
Id_main	Id_artis	Id_film	peran
M01	A01	F01	Toby Marshall
M02	A01	F02	Jesse Pinkman
M03	A02	F02	Walter White
M04	A03	F01	Dino Brewster

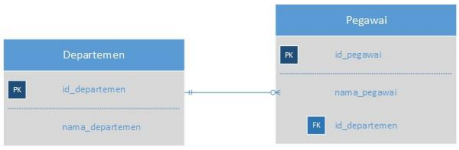
Gambar 11. Data tabel artis, film, main



Gambar 12. Hasil transformasi data dari Gambar 10

2) Transformasi dengan kardinalitas relasi antar entitas banyak ke satu atau satu ke satu

Gambar 13. ERD relasi banyak ke satu



Gambar 14. Tabel relasi entitas banyak ke satu

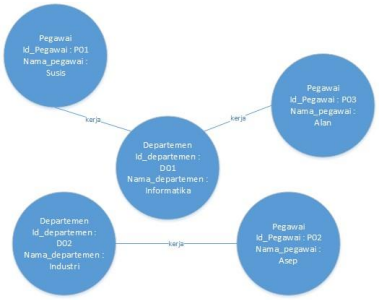
Tahapan transformasi untuk entitas dengan kardinalitas relasi banyak ke satu atau satu ke satu sebagai berikut :

1. ubah semua row pada tabel menjadi vertex dengan label vertex sama dengan nama tabel diikuti nilai setiap atribut pada row menjadi nilai properti pada vertex dengan catatan nilai null dan atribut yang menjadi foreign key tidak perlu dijadikan properti pada vertex, dan
2. Buat edge dengan label sesuai dengan relasi yang terbentuk antar dua entitas. Edge tidak memiliki properti karena tidak ada tabel yang menghubungkan kedua entitas.

Departemen	
Id_departemen	Nama_departemen
D01	Informatika
D02	Industri

Pegawai		
Id_pegawai	Nama_pegawai	Id_departemen
P01	Susis	D01
P02	Asep	D02
P03	Alan	D01

Gambar 15. Data departemen, pegawai



Gambar 16. Hasil transformasi data pada gambar 14

I. Uji Performansi

Pengukuran performansi pada kasus ini akan menggunakan dua parameter :

1. response time : lama waktu yang diperlukan oleh sistem untuk menyelesaikan suatu instruksi sejak instruksi tersebut dijalankan[10]. Pada kasus ini, response time adalah total penjumlahan execution time dan IO time, dan
2. query size : jumah vertex pada suatu graph query.

III. GAMBARAN SISTEM DAN SKENARIO PENGUJIAN

A. Gambaran Sistem

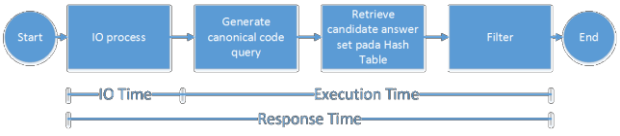
Sistem yang dibangun terdiri dari 2, yaitu graph database tanpa index dan graph database tanpa index. Adapun tahapnya sebagai berikut :

1. Transformasi data dari relational ke graph model,
2. Pembangunan sistem 1, yang mampu, melakukan pengambil data yang sudah ditransformasikan dan graph matching menggunakan algoritma graphQL[2],
3. Pembangunan sistem 2 dibangun dari sistem 1, yang telah ditambahkan fitur graph decomposition index.

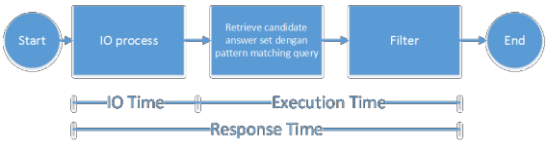
Sistem dibangun menggunakan java, dengan JDK 1.7 64bit dan dengan spesifikasi komputer yang memiliki Processor Intel(R) Core(TM) i7-3770M CPU @ 3.40Ghz (8 CPUs), ~3.4GHz danMemory 8192 MB RAM.

Pengukuran performansi pada kasus ini akan menggunakan dua parameter :

1. response time : lama waktu yang diperlukan oleh sistem untuk menyelesaikan suatu instruksi sejak instruksi tersebut dijalankan[10]. Pada kasus ini, response time adalah total penjumlahan IO time dan execution time. IO time adalah waktu yang diperlukan processor dalam memindahkan data dari hard disk ke main memory, sedangkan Execution time adalah lama retrieve data dari memory yang sesuai dengan query.
2. query size : jumah vertex pada suatu graph query.



Gambar 17. Ilustrasi pengukuran response time query pada Graph Database dengan Index

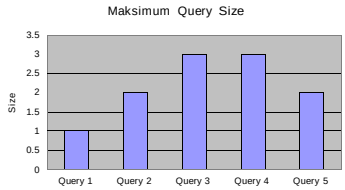


Gambar 18. Ilustrasi pengukuran response time query pada Graph Database dengan Index

B. Skenario Pengujian

Skenario pengujian yang diberikan kepada kedua sistem :

Query	Query
1	mencari visitor yang datang pada jam yang sama dengan mahasiswa yang memiliki nim 1103134332
2	mencari *co-author* dari buku yang ditulis oleh Tomassini, M.
3	mencari nama pengarang yang jumlah *copy*-an bukunya lebih dari 2
4	mencari nama pengarang yang jumlah *copy*-an bukunya lebih dari 2 dan pernah dipinjam oleh nim 112010128
5	buku yang hilang pada tahun 2013

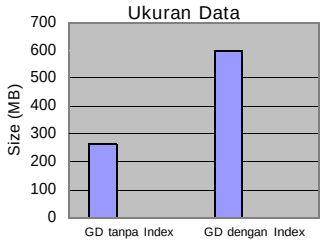


Gambar 19. Maksimum query size

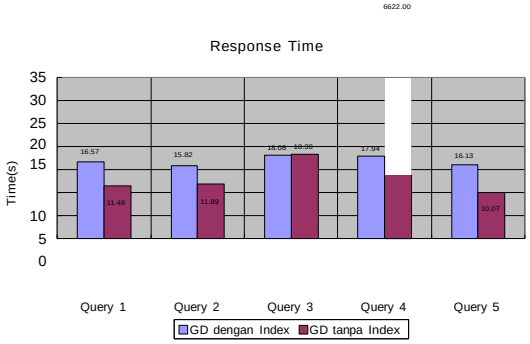
IV. HASIL PENGUJIAN DAN ANALISIS

A. HASIL PENGUJIAN

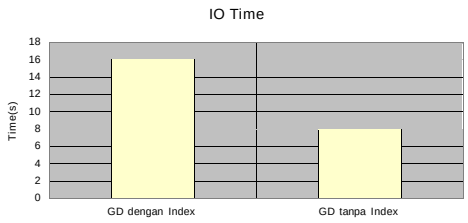
Berikut hasil pengujian pada kedua sistem :



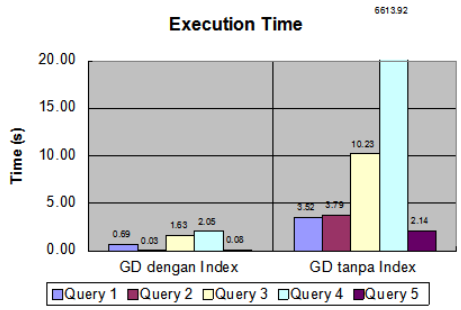
Gambar 20. Ukuran data pada kedua sistem



Gambar 21. Hasil Response Time dari pengujian kedua sistem



Gambar 22. Hasil IO Time dari pengujian kedua sistem



Gambar 23. Hasil Execution Time dari pengujian kedua sistem

B. ANALISIS

Berdasarkan hasil yang diperoleh, IO Time pada graph database tanpa index memiliki performa yang lebih bagus dibanding IO Time pada graph database dengan index. Hal ini terjadi karena ukuran data pada sistem graph database tanpa index lebih kecil dibanding ukuran data pada sistem graph database dengan index.

Seluruh hasil query menunjukkan bahwa execution time proses retrieve data pada graph database dengan index memiliki performa lebih baik dibanding execution time proses retrieve data pada graph database tanpa index. Hal ini terjadi karena untuk me-retrieve data pada graph database dengan index cukup dengan mengubah query menjadi canonical code dan mengakses hash table yang sesuai dengan canonical code yang dihasilkan, sedangkan pada graph database tanpa index proses retrieve dilakukan secara sequential ditambah lagi diperlukannya proses matching antara query dengan data yang ada.

Pada gambar 21 menunjukkan bahwa query size pada proses retrieve pada graph database tanpa index memiliki pengaruh besar pada lama execution time-nya. Hal ini dapat dilihat pada query 1, query 2, dan query 3.

Namun, pada query 3 dan query 4, serta query 1 dan query 5, hasilnya berbeda. Pada query 3 dan query 4 memiliki maksimum query size yang sama namun, hasil execution time-nya sangat jauh berbeda, begitu juga pada query 1 dan query 5 yang mana query 1 memiliki maksimum query size yang lebih kecil dari query 5 tetapi execution time-nya lebih lama. Setelah ditelusuri, ternyata ada faktor lain yang mempengaruhi execution time pada graph database tanpa index selain query size, yaitu jumlah kandidat data yang sesuai dengan query. Hal ini terkait dengan proses matching. Proses matching akan membentuk seluruh kemungkinan pola sesuai dengan kandidat data dan sesuai dengan query size. Pada query 3 dan query 4, query 4 menghasilkan kandidat data yang sesuai query lebih banyak dibanding pada query 3, sehingga execution time-nya lebih lama. Begitupun pada query 1 dan query 5. Oleh karena itu, proses matching memberi pengaruh besar pada execution time pada graph database tanpa index.

Sedangkan lama execution time pada graph database dengan index dipengaruhi besar oleh lama proses filter datanya dikarenakan tidak adanya proses matching seperti pada graph database tanpa index. Hal ini terlihat pada query 2 dan query 5, dimana kedua-duanya memiliki query size yang sama, tetapi hasil berbeda. Perbedaan itu disebabkan karena proses filter pada query 5 lebih kompleks dibanding proses filter pada query 2. Begitupun antara query 1 dengan query 2 dan query 3 yang mana query 1 memiliki query size yang lebih kecil tetapi memiliki execution time yang lebih lama.

Oleh karena itu, jika ditinjau dari execution time maka graph database dengan index memiliki performansi yang lebih bagus dibanding graph database tanpa index.

V. KESIMPULAN

Kesimpulan yang kami peroleh dari penelitian ini adalah pertama, Pada proses retrieve data, graph database dengan index memiliki performansi lebih bagus pada execution time dibanding graph database tanpa index karena pada graph database dengan index tidak terdapat proses matching sedangkan graph database dengan index sebaliknya. Kedua, peningkatan performansi dari execution time pada graph database setelah diterapkannya index memunculkan masalah baru yaitu graph database tanpa index memiliki IO time yang lebih bagus dibanding graph database dengan index karena ukuran data graph database dengan index lebih besar dibanding graph database tanpa index.

VI. DAFTAR PUSTAKA

[1] David W. Williams, et al, “Graph database indexing Using Structured graph decomposition”, University of North Caroline, Chapel Hill, 2007.
[2] Ee Jinsoo, et al, “An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases”, Kyungpook National University, Daegu, 2012.
[3] J. Huan, et al,“Comparing graph Representations of Protein Structure for Mining Family-Specific Residue-Based Packing Motifs” . Journal of Computational Biology(JCB), Vol. 12, No. 6, pp.657-671, 2005
[4] Kolosovskiy, M. A. (n.d.), “Data Structure for Representing a Graph: Combination of Linked List and Hash Table”, Altai State Technical University, Russia.
[5] Oracle, “HashMap Documentation”, http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html.
[6] Pfenning, F, “Lecture Notes on Hash Tables 15-122 : Principles of Imperative Computation”, September, 2010.
[7] Robinson, Ian, et al, “Graph databases”, Gravenstein. Highway North : O’Reilly, 2013.
[8] Robinson, Ian, et al, “Graph in a Relational World”, http://neo4j.com/graphacademy/online-course/.
[9] Rosen, K. H, “Discrete Mathematics and its Applications”, McGraw-Hill, New York, 2007.
[10] Sirait, Erika Christina, “Implementasi dan Analisis Performansi graph database pada Social Network Menggunakan Neo4j”, Institut Teknologi Telkom, Bandung, 2013.
[11] V, Z. L., “Lexicographic”, Chelyabinsk State University, Chelyabinsk, 2004.
[12] X. Yan, P. S. Yu, and J. Han, “Graph indexing : a frequent structure-based approach”, In Proceedings of the 2004 ACM SIGMOD international conference onManagement of data (SIGMOD’04), 2004.
[13] Silberschatz, A., et al, “Database System Concept, Fourth Edition”, New York, McGraw-Hill, 2001, ch. 12.