TUGAS BESAR 2
IF3140 Manajemen Basis Data
IMPLEMENTASI MEKANISME *CONCURRENCY CONTROL*

LAPORAN

Semester I Tahun Ajaran 2019/2020

Disusun oleh

13517010 Avisenna Abimanyu

13517043 Ihsan Imaduddin Azhar

13517073 Rayza Mahendra Guntara Harsono

13517106 Ikraduya Edian

13517136 Lucky Jonathan Chandra

TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2019

BAB I

DASAR TEORI

ACID (Atomicity Consistency Isolation Durability) merupakan empat hal yang perlu dijaga oleh sebuah basis data.

Atomicity
Atomicity berarti database memberikan kepastian bahwa antara seluruh transaksi berhasil dilaksanakan atau tidak sama sekali. Tidak boleh terjadi pengerjaan transaksi secara parsial yang hasilnya dilakukan *commit*.

Consistency
Consistency merupakan prinsip bahwa seluruh data pada database akan selalu konsisten. Data yang sudah terdapat pada database akan selalu valid dengan acuan aturan yang sudah ditetapkan, meliputi constraint, cascade dan trigger yang telah ditetapkan pada *database*.

Isolation
Isolation adalah jaminan bahwa semua transaksi terjadi tanpa mempengaruhi transaksi yang lain. Oleh karena itu transaksi tidak dapat membaca data dari transaksi lain yang belum selesai.

Durability
Durability berarti ketika transaksi sudah dilakukan dan di komit, transaksi itu akan tetap berada di sistem bahkan setelah terjadi crash pada sistem sekalipun. Semua perubahan pada sistem disimpan secara permanen. Ketika sistem memberitahu transaksi sukses kepada user, transaksi tersebut harus sukses seterusnya.

Untuk menjamin konsistensi dari basis data, dibutuhkan proses concurrency control untuk menjamin konsistensi perubahan data. Concurrency control dapat dilakukan dengan berbagai protokol. Protokol-protokol yang dapat diterapkan antara lain :

1. Lock-Based Protocols
2. Timestamp-Based Protocols
3. Validation-Based Protocols
4. Multiversion Schemes

Lock Based Protocol

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds

- Binary Locks − Lock pada data hanya ada dua state. Yaitu locked atau unlocked

- Shared/exclusive − Tipe ini menggunakan jenis tipe lock yang berbeda tergantung untuk apa digunakan. Jika menggunakan data untuk melakukan write operation, lock yang digunakan adalah exclusive lock. Mengizinkan dua atau lebih transaksi melakukan write akan mengakibatkan database yang inkonsisten. Read lock berbeda dengan write lock. Read lock merupakan shared lock karena tidak ada data yang mengalami perubahan

Timestamp Based Protocol

Protokol yang paling awam untuk dipakai adalah Timestamp Based Protocol

Perbedaan paling mencolok diantara kedua protokol adalah subjek dari protokol tersebut, lock based protocol menempatkan lock pada data transaksi sehingga variabel yang berubah dipengaruhi oleh operasi pada data. Timestamp Based Protocol melakukan penyimpanan *timestamp* dari awal transaksi tersebut dimulai sehingga operasi dilakukan bergantung pada transaksi itu sendiri.

Setiap transaksi ada timestamp yang diasosiasikan dan urutannya berdasarkan umur dari transaksi. Sebuah transaksi yang dilakukan pada 0002 akan lebih tua daripada semua transaksi yang terjadi setelahnya. Sebagai contoh transaksi ‘x’ dilakukan pada 0004 lebih muda dua sekon daripada transaksi yang dilakukan pada 0002

Sebagai tambahan, setiap data yang diberikan read dan write timestamp terakhir. Ini akan membuat sistem mengetahui kapan read dan write terakhir yang dioperasikan ke data item.

Validation Based Protocol

Pada optimistic *concurrency control*, tidak ada checking yang dilakukan saat transaksi sedang berlangsung. Pada skema ini, update dari transaksi tidak akan dilakukan *commit* sampai transaksi berakhir. Saat transaksi dieksekusi, semua update akan diaplikasikan pada salinan lokal pada item dari data yang disimpan untuk transaksi. Pada akhir eksekusi, fase validasi akan mengecek apakah ada transaksi yang melanggar kaidah serializable. Jika tidak dilanggar, transaksi akan di commit dan database di update dengan salinan lokal. Sebaliknya, jika melanggar, transaksi akan diulang nanti.

Ada tiga fase pada concurrency control protocol:

1. Read phase: Suatu transaksi dapat membaca nilai dari suatu data yang sudah di-*commit* dalam *database*. Namun seluruh update hanya dilakukan pada salinan lokal dari data yang berada di basis data yang berada di tempat transaksi dijalankan.

2. Validation phase: Pengecekan dilakukan untuk memastikan serializability tidak dilanggar jika update transaksi dilakukan pada basis data.

3. Write phase: Jika fase validasi berhasil, update transaksi akan dilakukan pada database. Update akan dihapus jika transaksi diulang.

Multiversion concurrency control (**MCC** or **MVCC**),

Multiversion concurrency control adalah metode concurrency control yang digunakan oleh banyak database management system yang menerapkan akses secara konkuren ke database dan pada bahasa programming untuk menerapkan memori transaksi. Ketika MVCC database harus update sebuah data, tidak akan menimpa data item dengan data baru. Tapi yang dilakukan adalah membuat versi baru dari data item. Oleh karena itu ada banyak versi yang disimpan.

MVCC menggunakan timestamps (**TS**), dan *incrementing transaction IDs* untuk mencapai *transaction consistency*. MVCC membuat sebuah transaksi tidahk harus menunggu untuk melakukan read pada object database dengan membuat banyak versi dari sebuah object. Setiap version dari object mempunyai *Read Timestamp* (RTS) dan *write timestamp* (WTS) yang membuat sebuah transaksi membaca versi paling terakhir jika ingin membaca sebuah object.

Pada time=1 status dari database adalah:

Time	Object 1	Object 2
0	"Foo" by T0	"Bar" by T0
1	"Hello" by T1	

T0 menulis Object 1="Foo" dan Object 2="Bar". Lalu T1 menuliskan Object 1="Hello" dan menempatkan object 2 dengan nilai originalnya. Nilai pada object 1 akan menimpakan nilai objek saat Time = 0 untuk seluruh transaksi setelah T1 selesai objek1 akan dibersihkan.

Bila transaksi T2 yang sudah berjalan lama melakukan operasi read pada Object 2 dan Object 2 setelah T1 dilakukan commit dan ada transaksi update di T3 yang menghapus Object 2 dan menambah Object 3="Foo-Bar",state databasenya saat Time = 2 akan menjadi:

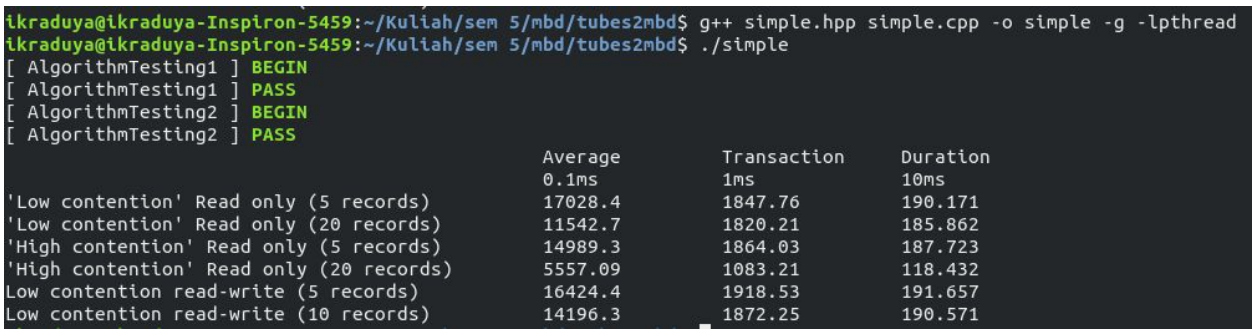
Time	Object 1	Object 2	Object 3
0	"Foo" by T0	"Bar" by T0	
1	"Hello" by T1		
2		(deleted) by T3	"Foo-Bar" by T3

Akan ada versi time 2 dari Object 2 yang ditandai sebagai sudah dihapus dan Object 3 yang baru, karena T2 dan T3 dijalankan secara concurrent, T2 menafsirkan versi database sebelum Time = 2 sehingga T2 reads Object 2="Bar" dan Object 1="Hello". Inilah bagaimana MVCC melakukan concurrency control tanpa adanya lock.

BAB II
Algoritma dan Analisis

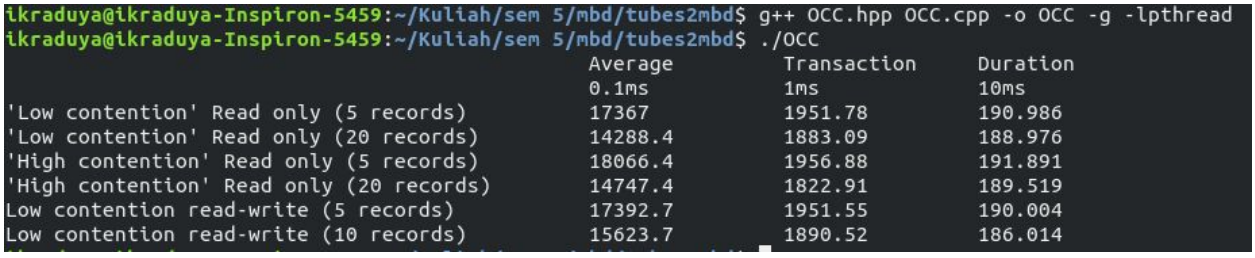
Untuk setiap algoritma, sertakan :

1. Screenshot hasil percobaan
   a. Simple Locking (Exclusive Only)



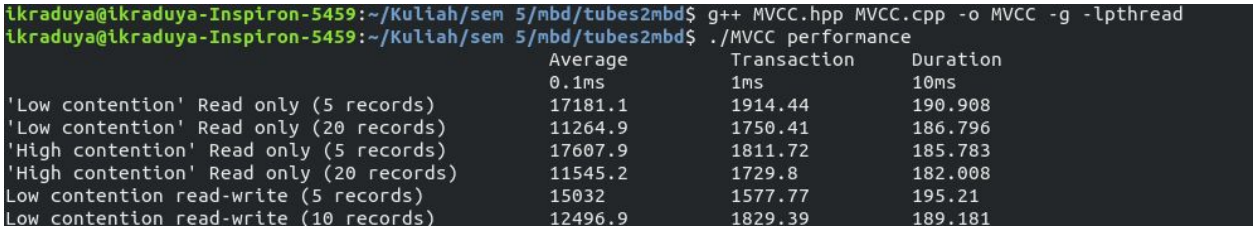
```
ikraduya@ikraduya-Inspiron-5459:~/Kuliah/sem 5/mbd/tubes2mbd$ g++ simple.hpp simple.cpp -o simple -g -lpthread
ikraduya@ikraduya-Inspiron-5459:~/Kuliah/sem 5/mbd/tubes2mbd$ ./simple
[ AlgorithmTesting1 ] BEGIN
[ AlgorithmTesting1 ] PASS
[ AlgorithmTesting2 ] BEGIN
[ AlgorithmTesting2 ] PASS
                                        Average         Transaction     Duration
                                        0.1ms           1ms             10ms
'Low contention' Read only (5 records)  17028.4         1847.76         190.171
'Low contention' Read only (20 records) 11542.7         1820.21         185.862
'High contention' Read only (5 records) 14989.3         1864.03         187.723
'High contention' Read only (20 records)5557.09         1083.21         118.432
Low contention read-write (5 records)   16424.4         1918.53         191.657
Low contention read-write (10 records)  14196.3         1872.25         190.571
```

   b. Optimistic Concurrency Control



```
ikraduya@ikraduya-Inspiron-5459:~/Kuliah/sem 5/mbd/tubes2mbd$ g++ OCC.hpp OCC.cpp -o OCC -g -lpthread
ikraduya@ikraduya-Inspiron-5459:~/Kuliah/sem 5/mbd/tubes2mbd$ ./OCC
                                        Average         Transaction     Duration
                                        0.1ms           1ms             10ms
'Low contention' Read only (5 records)  17367           1951.78         190.986
'Low contention' Read only (20 records) 14288.4         1883.09         188.976
'High contention' Read only (5 records) 18066.4         1956.88         191.891
'High contention' Read only (20 records)14747.4         1822.91         189.519
Low contention read-write (5 records)   17392.7         1951.55         190.004
Low contention read-write (10 records)  15623.7         1890.52         186.014
```

   c. Multiversion Timestamp Concurrency Control



```
ikraduya@ikraduya-Inspiron-5459:~/Kuliah/sem 5/mbd/tubes2mbd$ g++ MVCC.hpp MVCC.cpp -o MVCC -g -lpthread
ikraduya@ikraduya-Inspiron-5459:~/Kuliah/sem 5/mbd/tubes2mbd$ ./MVCC performance
                                        Average         Transaction     Duration
                                        0.1ms           1ms             10ms
'Low contention' Read only (5 records)  17181.1         1914.44         190.908
'Low contention' Read only (20 records) 11264.9         1750.41         186.796
'High contention' Read only (5 records) 17607.9         1811.72         185.783
'High contention' Read only (20 records)11545.2         1729.8          182.008
Low contention read-write (5 records)   15032           1577.77         195.21
Low contention read-write (10 records)  12496.9         1829.39         189.181
```

2. Hasil analisis dari algoritma yang diterapkan

Protokol	Average Transaction Duration		
	0.1ms	1ms	10ms
	Throughput		
Simple Locking	13289.70	1734.33	177.40
OCC	16247.6	1909.46	189.565
MVCC	14188	1768.92	188.31

Untuk setiap protokol concurrency, dijalankan enam jenis testing dengan penjelasan sebagai berikut:

a. ‘Low Contention’ Read only (5 records)
   Data item yang tersedia sebanyak 1000000, 5 operasi read untuk masing-masing transaksi.
b. ‘Low Contention’ Read only (20 records)
   Data item yang tersedia sebanyak 1000000, 20 operasi read untuk masing-masing transaksi.
c. ‘High Contention’ Read only (5 records)
   Data item yang tersedia sebanyak 100, 5 operasi read untuk masing-masing transaksi.
d. ‘High Contention’ Read only (20 records)
   Data item yang tersedia sebanyak 100, 20 operasi read untuk masing-masing transaksi.
e. ‘Low Contention’ Read-Write only (5 records)
   Data item yang tersedia sebanyak 1000000, 5 operasi read kemudian write untuk masing-masing transaksi.
f. ‘Low Contention’ Read-Write only (10 records)
   Data item yang tersedia sebanyak 1000000, 10 operasi read kemudian write untuk masing-masing transaksi.

Jumlah transaksi yang dijalankan secara konkuren diatur sebanyak 100.

Pada tabel di atas per baris protokol concurrency control yang dieksplorasi, didapatkan rata-rata throughput koresponden dengan average transaction duration tertentu. Throughput adalah jumlah transaksi dibagi durasi penyelesaian contohnya pada protokol simple locking throughput ketika average transaction duration 0.1ms, 1ms, 10ms masing-masing 13289.70, 1734.33 dan 177.40. Hal ini menyatakan semakin kecil average transaction duration, yang menandakan semakin tinggi konkurensi, throughput yang didapat lebih tinggi, dengan kata lain performa algoritma lebih tinggi.

Berdasarkan hasil eksperimen, pada protokol simple locking diperoleh nilai throughput 13289.70, 1734.33, dan 177.40. Untuk protokol OCC diperoleh nilai throughput 16247.6, 1909.46, dan 189.565. Terakhir untuk protool MVCC diperoleh nilai throughput 14188, 1768.92, dan 188.31. Masing-masing nilai throughput tersebut berkorespondensi dengan average transaction duration dengan nilai 0.1, 1, dan 10 milisekon.

BAB III

Perbandingan Algoritma

Menurut hasil eksperimen yang kami lakukan. Algoritma OCC merupakan algoritma yang mempunyai *throughput* tertinggi dibandingkan dengan algoritma yang menggunakan protokol MVCC dan simple locking. Throughput yang dimiliki algoritma OCC pada durasi waktu 10 ms, 1 ms, dan 0.1 ms bernilai lebih besar jika dibandingkan dengan kedua algoritma lainnya. Sedangkan algoritma Simple locking merupakan algoritma yang mempunyai *throughput* paling rendah jika dibandingkan dengan 2 algoritma protokol konkurensi lainnya. Nilai throughput algoritma simple locking pada durasi waktu 10 ms, 1 ms, dan 0.1ms bernilai lebih kecil jika dibandingkan dengan 2 algoritma lainnya.

Dari ketiga protokol konkurensi, nilai throughput pada waktu 0.1 milisekon lebih besar daripada nilai throughput pada waktu 1 milisekon. Demikian pula nilai throughput pada waktu 1 milisekon lebih besar daripada nilai throughput pada waktu 10 milisekon. Average transaction time merupakan rata-rata waktu yang diperlukan untuk menyelesaikan transaksi. Semakin kecil waktunya makan perpindahan eksekusi antar transaksinya semakin sering terjadi. Sehingga tingkat konkurensinya tinggi. Begitupun sebaliknya jika nilai rata-rata waktu yang diperlukan cukup besar. Maka perpindahan eksekusi antar transaksi dalam periode waktu tertentu tidak banyak, sehingga tingkat konkurensinya rendah.

BAB IV

Kesimpulan dan Saran

Jenis protokol konkuren yang memiliki performa terbaik merupakan protokol konkuren OCC dengan throughput 16247.6, 1909.46, dan 189.565 untuk average transaction duration 0.1, 1, dan 10 milisekon. Dan protokol konkuren yang memiliki performa terburuk merupakan tipe simple locking dengan throughput 13289.7, 1734.33, dan 177.4 untuk average transaction duration 0.1, 1, dan 10 milisekon.

Dalam penyelesaian tugas besar ini, tim kami menemui kesulitan dalam pengembangan program. Kesulitan yang dialami seperti *memory leak* dalam program, pengembangan *framework*, dan pengembangan kode program. Sebaiknya dalam mengembangkan program yang kompleks, kinerja anggota tim diefisienkan dengan menggunakan *tools* dan *extension* yang memudahkan anggota seperti *extension live share* di IDE dan menggunakan

BAB V
Pembagian Kerja

Nama	Pembagian kerja	Persentase (%)
Avisenna Abimanyu	OCC	20
Ikraduya Edian	MVCC	20
Rayza Mahendra	OCC	20
Ihsan Imaduddin Azhar	Simple Locking	20
Lucky Jonathan Chandra	MVCC	20

BAB VI

Referensi

https://github.com/dhatch/database-concurrency-control.
https://www.geeksforgeeks.org/timestamp-based-concurrency-control/
https://en.wikipedia.org/wiki/Timestamp-based_concurrency_control
https://en.wikipedia.org/wiki/Lock-based_concurrency_control
https://vladmihalcea.com/how-does-mvcc-multi-version-concurrency-control-work/