Analisis Arsitektur Perancangan Sistem Terdistribusi Menggunakan Metode *Nested Transaction* pada Lingkungan Kerja Perkantoran

Antonius Bima Murti Wijaya[1], Jeffrey Soeprapto[2], Yusuf Tegar[3]
Program Studi teknik Informatika, Fakultas Teknologi Industri, Universitas Atma Jaya Yogyakarta, Jl Babarsari No. 43 Yogyakarta 55281
Email: [1]bimamurti@gmail.com, [2]jeffrey.soeprapto@gmail.com, [3]tegar_chen@hotmail.com

Abstract

The length of integration proses and the extinction of that effort describe the level of difficulty or complexity of business integration. The difficulties whic are experienced are not only in terms of logical but also the physical side. If the physical design of the architecture is not good, there will be a loss in performance because the process of each transaction will be executed sequentially. Similarly, the high computational load in a computer because of some new functions that have the wrong placement in the architecture and design can cause centralized computing. Nested transaction method that can be paralized data transaction will be used to spread computation, and using the combination between database fragmentation and replication will create a faster, effective and efficient system.

Keywords—*Distributed database, database, nested transaction*

1. PENDAHULUAN

Perkembangan teknologi sistem informasi yang semakin pesat ini membuat setiap perusahaan dituntut untuk mendapatkan informasi informasi penting yang berasal dari berbagai department yang dimilikinya. Oleh karena itu diperlukan adanya gabungan informasi antara data-data yang ada pada setiap department. Beragam tuntutan yang bermuara pada keinginan untuk ”mengintegrasikan” secara fisik maupun relasi dua atau lebih organisasi tersebut bermuara pada kebutuhan melakukan upaya ”sharing” sejumlah sumber daya data dan informasi yang dimiliki sesama organisasi. Dengan kata lain diperlukan adanya integrasi antara setiap sistem informasi yang ada pada setiap department. Dalam konteks ini tidak lah semudah kelihatannya untuk melakukan integrasi terhadap masing masing aplikasi. Lamanya proses integrasi dan sering kandasnya usaha tersebut menggambarkan tingkat kesulitan atau kompleksitas usaha integrasi yang dimaksud. Dengan muncul nya Integrasi antara sistem dimungkinkan untuk munculnya suatu proses Kebutuhan baru, dan sering kali kebutuhan ini tidak mampu ditangani oleh salah satu sistem informasi. Begitulah gambaran seberapa pentingnya integrasi sistem yang terdistribusi namun sangat berlawanan dengan tingginya tingkat kesulitan implementasi. Kesulitan yang dialami juga tidak hanya dari sisi logic saja namun juga sisi fisiknya. Jika rancangan fisik arsitekturnya tidak baik maka akan terjadi penurunan performa karena proses tiap transaksi akan mengalami eksekusi secara sequential. Begitu pula tingginya beban komputasi dalam satu komputer karena adanya beberapa fungsi fungsi baru yang memiliki penempatan yang salah dalam arsitektur dan perancangannya sehingga menimbulkan komputasi terpusat.

Jurnal ini, akan membahas perancangan model-model arsitektur sistem tersistribusi untuk perusahaan secara global. Metode yang dipakai hanya menggunakan metode *nested transaction* yang akan dipadukan dengan replika dan fragmentasi.

Urutan pembahasan adalah Pendahuluan yang berisi paparan singkat pentingnya permasalahan yang dibahas. Tinjauan Pustaka yang berisi pendeskripsian masalah-masalah sistem terdistribusi dengan menggunakan *nested transaction.* Contoh kasusnya berisi pembagian tugas-tugas sistem terdistribusi dari perusahaan secara global yaitu *back-end* (misal personalia) dan *front-end (*misal transaksi dan registrasi). Pembahasan berisi bagaimana cara mengatasi masalah sistem terdistribusi yang terjadi di perusahaan secara global. Penutup yang merupakan kesimpulan dari keseluruhan masalah.

2. Tinjauan Pustaka

Pada era sekarang, banyak sistem terdistribusi yang menggunakan *nested transaction* karena dapat memperlancar transaksi dengan mengefisiensikan pengiriman data. Efisiensi pengiriman data dilakukan dengan membagi pengiriman data pada *server* sehingga kinerja *server* dapat bekerja secara merata. *Nested transaction* sangat berguna untuk mengefisiensikan pengirimian data namun juga memiliki masalah-masaha tertentu.

Untuk memanfaatkan potensi yang maksimal dari *Nested transaction* maka memerlukan butir-butir kontrol konkurensi yang cocok serta metode akses untuk berbagi data. Metode yang digunakan adalah downward inheritance of locks untuk membuat data yang dimanipulasi oleh parents yang tersedia untuk childred[1]. Hasilnya downward inheritance of locks dapat memanipulasi oleh parents yang tersedia untuk children.

Selain itu juga terdapat kesulitan pada implementasi *nested transaction* ke perangkat keras. Metode untuk mengatasi masalah ini dengan menggambarkan semantic-semantic untuk serializable *nested transaction*. Metode ini dibuat oleh [2] sehingga masalah ini dapat dipecahkan. Setelah melakukan penerapan ini maka akan lebih mendukung perangkat keras dari kasus umum. *Nested transaction* membutuhkan solusi untuk memeriksa referensi integrity constraints dan memiliki hambatan global di dalam sistem relational multi *database*.Cara untuk mengatasi masalah ini dengan mengimplementasi lebih dari satu PC cluster dengan menggunakan Oracle9i DBMS [3]. Setelah melakukan penerapan ini maka pengukuran waktu yang dibutuhkan untuk memeriksa kendala global di dalam *distributed systems.* Dan menghasilkan biaya overhead bisa turun sebanyak 50%. Selain itu penempatan memory pada metode *nested transaction* biasa masih kurang efektif, oleh karena itu untuk memaksimalkan kinerja metode ini diperlukan pengembangan ke arah open *nested transaction*[4], keefisiansian pada *nested* transaksi masih juga bisa ditingkatkan pada bagian penjadwalannya yaitu dengan menggunakan *flexible high reward for nested transactions*[5]

Dengan sistem terdistribusi yang memiliki banyak *database* seperti ini, kesulitan utama yang terjadi adalah mengenai sinkronisasi data. Untuk itu diperlukan hirarki dan flat protokol yaitu 2PC-RT-NT dan mekanisme kunci yaitu 2LP-NT-HP[6] sehingga dapat teratasi untuk masalah sinkronisasi data dan juga mencegah konflik data pada sistem *nested transaction*.

Pada jurnal ini akan membahas tentang metode *nested transaction* yang diterapkan pada proses bisnis perusahaan secara global dan bagaimana cara mengkombinasikannya dengan beberapa metode yang lain yang berguna untuk mendukung pemerataan komputasi. Berbeda dengan jurnal jurnal lain yang membahas kepada pokok permasalahan *nested transaction* saja tapi tidak tentang kombinasi kombinasi metodenya.

2. Landasan Teori

2.1 Sistem Terdistribusi

Sistem terdistribusi adalah suatu sistem (*hardware + software*) yang terdiri atas kumpulan dari komputer yang saling terhubung dan terlihat bagi pengguna sistem sebagai suatu sistem komputer tunggal. Sistem terdistribusi dilakukan oleh suatu pihak karena membutuhkan sharing sumber daya, pemerataan sumber daya, replikasi data untuk *fault-tolerance,* Integrasi antara sistem lama dan sistem baru, kebutuhan akan sistem yang memiliki mobilitas tinggi seperti laptop. Berikut merupakan karakteristik dari sebuah sistem terdistribusi:

1. *Resource Sharing*: kemampuan untuk menggunakan hadware, software dan data dimanapun di dalam sistem.
2. *Openess*: kemampuan sistem untuk mengadopsi serta mengintegrasikan berbagai standar, teknologi dan sistem yang ada.
3. *Concurrency*: kemampuan untuk menjalankan komponen-komponen dalam sistem sebagai proses yang konkuren.
4. *Fault tolerance*: kemampuan sistem untuk menangani secara tuntas segala macam kesalahan yang mungkin terjadi
5. *Transparency*: Yaitu kemampuan sistem untuk membuat sistem terlihat sebagai sistem yang monolitik bagi perspektif pengguna dan mungkin juga bagi perspektif programmer

2. 2 Transaksi data

Nested Transaction adalah sebuah transaksi dimana setiap transaksi utama dapat membuka subtransaksi dan setiap subtransaksi dapat membuka subtransaksi lagi hingga beberapa tingkatan. Kelebihan dari *nested transaction* adalah mampu melakukan eksekusi transaksi secara pararel dan tidak secara sequential sehingga lalu lintas jaringan bisa menjadi lebih teratur.

Flat transaction adalah suatu adalah suatu seri operasi yang dibentuk secara inti atomik yang merupakan suatu untuk kerja sendiri. *Flat transaction* yang paling dasar bisa dilakukan dengan mengakaitkan 1 *server* aplikasi dengan satu *database*.

2. 3 Fragmentasi

Fragmentasi merupakan sebuah proses pengambilan bagian-bagian baris ataupun kolom dari table-tabel sebagai unit data terkecil yang akan dikirimkan melalui jaringan computer. Sedangkan fragmentasi data merupakan proses dimana basis data akan dipecah-pecah kedalam unit-unit logic yang disebut fragment yang kemudian akan disimpan dalam site yang berbeda, fragmentasi data tergabung dalam proses desain basis data, yang kemudian akan menentukan apa yang kemudian akan dilakukan terhadapa fragment yang telah dibuat.

Fragmentasi Horizontal

Fragmentasi horizontal terdiri atas tuple. Fragmentasi model ini sangat berguna didalam basis data terdistribusi dimana setiap tuple dapat berisi data yang memiliki properti secara umum.

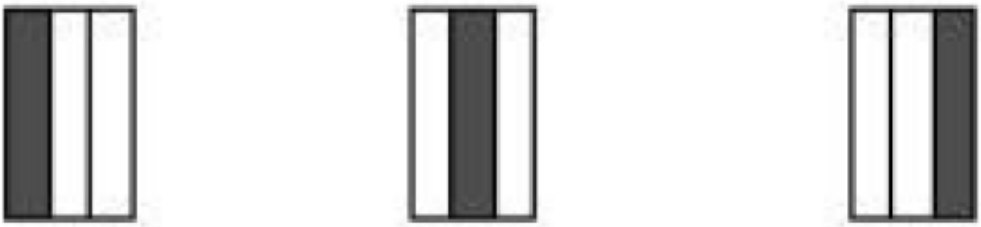
Gambar 1 bentuk fragmentasi horizontal

Kemudian fragmentasi horizontal dibagi kedalam 2 bagian, yaitu :

1. *Primary* horizontal *fragmentation*
   Ketika fragmentasi horizontal terjadi melalui predikat yang dapat secara langsung diaplikasikan pada tuple dari relasi
2. *Derrived* horizontal *fragmentation*
   Ketika fragmentasi horizontal terjadi dari sebuah predikat yang diaplikasikan pada tuple dalam relasi yang berbeda. *(Foreign Key)*

Fragmentasi *Vertical*

Fragmentasi vertikal akan membagi lagi atribut-atribut dari table yang tersedia menjadi beberapa grup. Bentuk yang paling simple dari fragmentasi ini adalah dekomposisi, dimana sebuah row id yang unik dapat disertakan dalam setiap fragment untuk menjamin dan menjaga adnya kemungkinan terjadinya proses rekonstruksi melalui sebuah operasi join.



Gambar 2 bentuk fragmentasi vertikal

Fragmentasi *Mixed Atau Hybrid*

Fragmentasi jenis ini merupakan gabungan dari penggunaan kedua proses fragmentasi diatas. Memiliki 2 jenis yaitu :

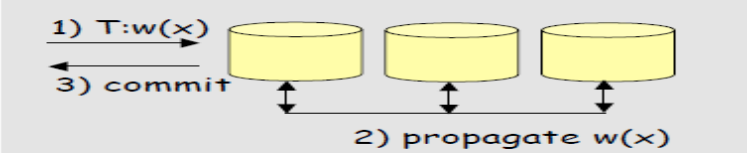
1. Fragmentasi *Horizontal* pada Fragmentasi *Vertical*
2. Fragmentasi *Vertical* pada Fragmentasi *Horizontal*

2. 4 Replikasi

Replikasi adalah suatu teknik untuk melakukan copy dan pendistribusian data dan objek-objek *database* dari satu *database* ke *database* lain dan melaksanakan sinkronisasi antara *database* sehingga konsistensi data dapat terjamin. Dengan menggunakan teknik replikasi ini, data dapat didistribusikan ke lokasi yang berbeda melalui koneksi jaringan lokal maupun internet. Replikasi juga memungkinkan untuk mendukung kinerja aplikasi, penyebaran data fisik sesuai dengan penggunaannya seperti pemrosessan *database* terdistribusi melalui beberapa *server*.

Berdasarkan kapan data diupdate :

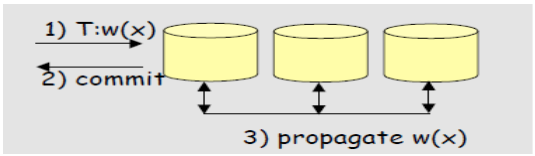
a. Synchronous



Gambar 3 Replikasi Sinkronus

1. Terjadi sebuah transaksi pada sebuah *client*.
2. Permintaan transaksi tadi dikirim ke *client* lainnya lalu dikumpulkan.
3. Setelah semua *client* lain mengumpulkan, baru dilakukan respon ke *client* yang meminta transaksi tadi.

b. Asynchronous

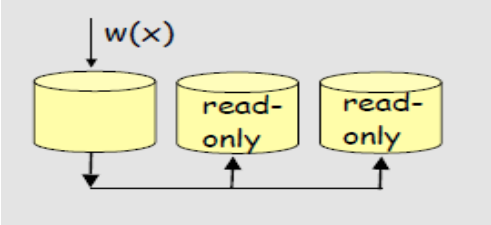


Gambar 4 Replikasi Asinkronus

1. Terjadi sebuah transaksi pada sebuah *client*.
2. Permintaan transaksi tadi dikirim ke *client* lainnya lalu langsung memberikan respon ke *client* yang merequest tadi.
3. *Client* yang merequest mengeksekusi 1 per 1 hasil respon dari *client* lainya tadi.

Berdasarkan dimana data diupdate :

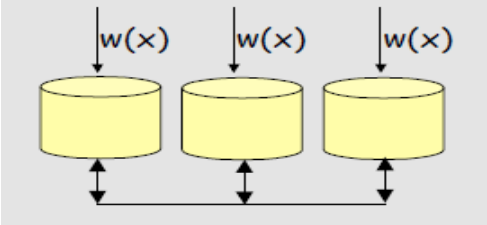
a. Primary Copy (master)



Gambar 5 Primary Copy

Dari gambar di atas menunjukan bahwa pemrosesan hanya dilakukan pada basisdata master. Sedangkan lainnya hanya sebagai slave. Pada basisdata slave hanya melakukan pembacaan data, sedangkan proses mengubahan hanya terjadi pada master yang diikuti oleh slave.

b. Update Everywhere



Gambar 6 Update Everywhere

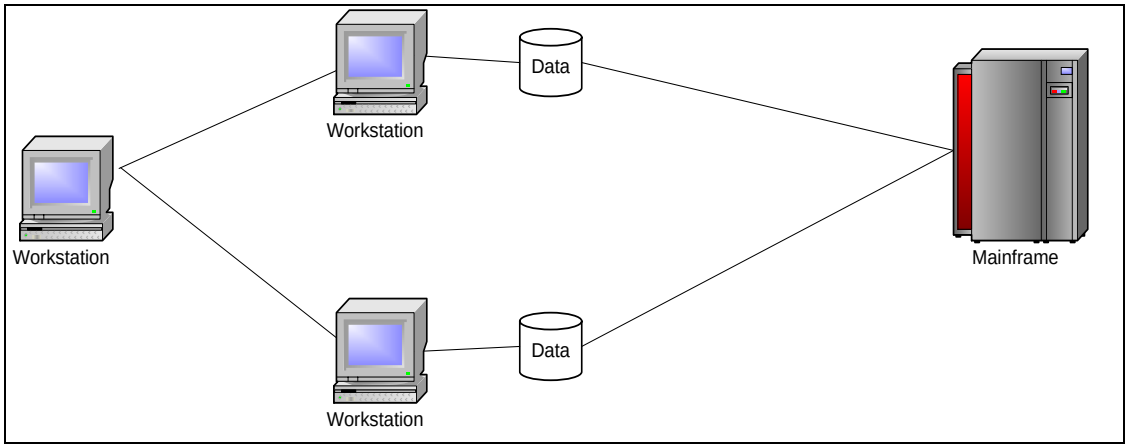
Dari gambar di atas menunjukan bahwa pemrosesan dilakukan di setiap basisdata. Proses pengubahan isi basisdata dapat dilakukan di setiap basisdata yang ada.[7]

3 Penjabaran Masalah

Pada sistem perkantoran perusahaan terdapat berbagai macam department. Masing- masing departmen memiliki proses bisnis yang berbeda-beda. *Requirement* yang dibutuhkan untuk suatu sistem pun berbeda. Misalnya saja pada bagian department Keuangan dibutuhkan sistem untuk melakukan perhitungan perhitungan debet dan kredit, sementara pada bagian *Advertising* diperlukan sistem untuk menangani pencatatan wilayah wilayah tempat promosi, dan pada bagian *Marketing* diperlukan untuk menangani pencatatan wilayah wilayah pemasaran. Dari berbagai macam requirment ini akan sangatlah sulit untuk disatukan kedalam satu sistem informasi. Namun dari sisi lain, berbagai department ini memiliki *requirment-requirment* yang sama misalnya saja *requirment* untuk melihat gaji masing-masing pegawai, melihat perkembangan perusahaan, login kedalam sistem, dan lain lain tergantung dari proses bisnis tiap tiap perusahaan. Permasalahan ini akan bertambah sulit ketika data yang masuk ke *database* ada dalam jumlah banyak dan berkelanjutan, misalnya seperti proses transaksi dimana hampir setiap waktu ada perubahan pada *database*-nya yang mengakibatkan komputasi yang tinggi. Jika kesemua proses ditangani secara terpusat, beban komputasi dan kesibukan jaringan yang tinggi akan terjadi pada *server*. Oleh karena itu diperlukan suatu sistem yang tersebar begitu pula dengan *database*nya sehingga beban komputasi dan kesibukan jaringan bisa disama-ratakan, sehingga dapat meningkatkan kecepatan akses terhadap *database*.

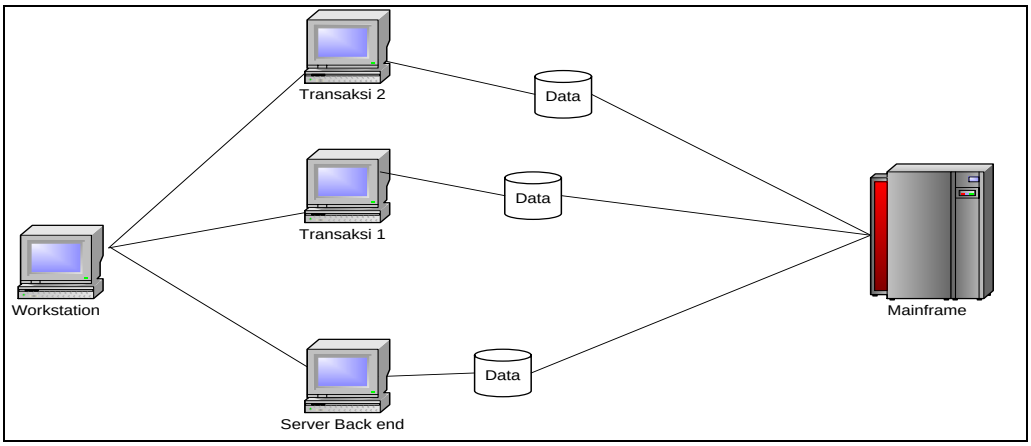
4. PEMBAHASAN

Untuk melakukan perancangan arsitekturnya yang rinci tidak bisa diterapkan secara general, sebab setiap perusahaan pun memiliki proses bisnis yang berbeda-beda, oleh karena itu cukup melihatnya dari aspek aspek umum yang dimiliki perusahaan. Proses perusahaan dibagi menjadi 2 bagian yaitu proses *front-end dan back-end*. Proses *back-end* merupakan proses proses yang berkaitan dengan kegiatan internal perusahaan, sementara proses *front-end* merupakan proses yang berkaitan dengan kegiatan perusahaan ke pelanggan. Pada proses *back-end* akan menyimpan data-data pribadi dan private perusahaan seperti data pekerja, department, jadwal, dsb, sementara pada *front*-end perusahaan, menyimpan data data yang berkaitan langsung dengan pelanggan dan selalu berubah ubah dengan frekuensi yang tinggi misal data transaksi, dan data pelanggan. Pada beberapa perusahaan ada yang menyatukan kedua data ini kedalam satu *database server* namun ada juga yang memisahnya, alasannya pun beraneka ragam mulai dari tingkat keamanan, sampai mungkin terjadinya interfensi. Jika proses *front-end* ini begitu sibuk lalu lintas datanya, maka perlu juga dilakukan pendistribusian sistemya. Dimana ada satu komputer sendiri yang menangani proses transaksi ataupun proses *front-end* perusahaan sementara proses *back-end* dilakukan pada komputer lain.



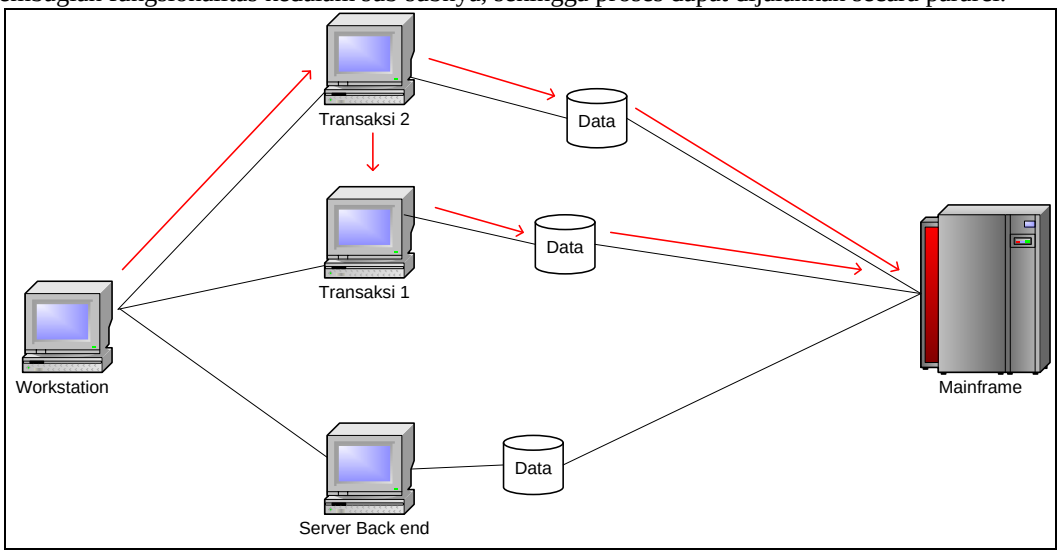
Gambar 7 Model 1

Dari gambar dapat dilihat bahwa ketika suatu komputer atau user sistem ingin mengakses data *back-end* perusahaan akan masuk pada *server* yang berbeda dengan ketika ingin mengakses data *front-end* perusahaan, dengan begini komputasi akan terpisah menjadi 2 tempat yang berbeda. Dan komputer utama (*mainframe*) hanya bertugas untuk menyalurkan data ke *database database* anaknya. Dengan begini waktu tunggu yang dibutuhkan untuk melakukan akses data akan lebih berkurang. Untuk melakukan support dengan performa kerjanya dapat dilakukan dengan replikasi *database*. Biasanya pada bagian *front-end* perusahaan masih terdapat banyak proses transaksi yang mengakibatkan kemacetan pada sisi *server front-end*, contohnya pada perbankan. Oleh karena itu diperlukan adanya pemisahaan sistem kembali.



Gambar 8 Model 2

Pada bentuk arsitektur lainnya tersebut, bagian *front-end* dipisahkan lagi sehingga ketika *server* transaksi 1 sedang sibuk lalulintasnya, maka proses dapat dialihkan ke transaksi 2. Atau juga bisa transaksi 1 dan transaksi 2 dipisah berdsarkan fungsionalitasnya, misal pada perbankan transaksi 1 untuk proses penarikan uang dan transaksi 2 untuk proses pengiriman uang. Secara sederhana ini bisa disebut sebagai *nested transaction* karena ada nya pembagian fungsionalitas kedalam sub subnya, sehingga proses dapat dijalankan secara pararel.

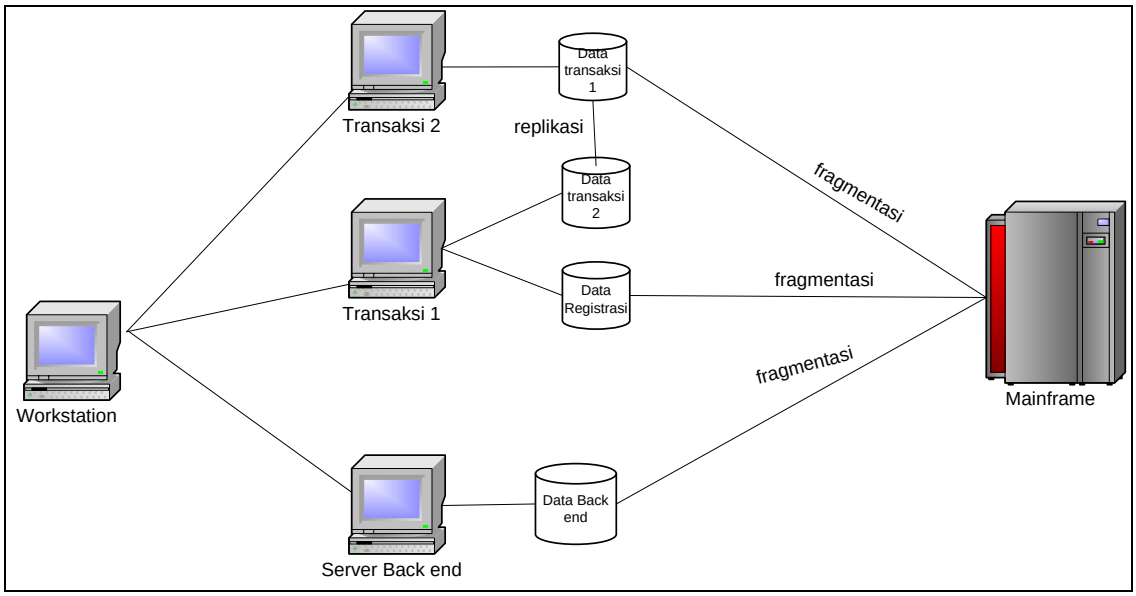


Gambar 9 Model 3

Model 3 adalah contoh arsitektur dimana bagian *front-end* untuk transaksi sedang sibuk sehingga proses transaksi ditangani oleh dua *server* yaitu *server* transaksi 1 dan *server* transaksi 2. *Server* utama tetap pada *server* transaksi 1 sedangkan *server* transaksi 2 sebagai *server* pembantu porses transaksi , dengan demikian proses transaksi dapat berjalan lebih baik dan lebih cepat dalam proses transaksi.

Dalam proses yang terdistribusi ini memiliki kelemahan yaitu tingkat kesulitan yang tinggi untuk melakukan sinkronisasi data, karena *data base* tidak hanya terdapat pada *server* pusatnya namun juga terdapat pada *server server* cabang nya. Dimana setiap informasi yang masuk harus segera dikirim ke *server* pusat untuk peng-update-an sementara tidak boleh mengganggu proses lainnya, sistem harus mampu kapan proses transaksi yang terjadi menurun, sehingga sumber daya nya bisa dipakai untuk pengiriman data ke *server* pusat.

Banyak Pemisahan/ pendistribusian sistem ini akan dilakukan seiring dengan banyaknya proses yang terjadi atau dengan kata lain proses yang melibatkan banyak pemakai dan disesuaikan dengan kebutuhan. Kesemuanya ini dilakukan untuk mencegah terjadinya komputasi terpusat yang dapat mengakibatkan *bottleneck effect* dimana banyak permintaan pada suatu *server* namun daya tampung yang dimiliki *server* tidak mencukupi sehingga mengakibatkan terjadinya antrian.



Gambar 10 Model 4

Model 4 ini merupakan model untuk mengefektifkan komputasi dengan menggabungkan fragmentasi *database* dengan replikasi. Model 4 ini merupakan perwujudan dari model 3. *Database* untuk *server front-end* dan *back-end* pemetaan *database*nya menggunakan fragmentasi agar tidak ada interferensi antara *database front-end* dan *back-end*. Fragmentasi pada data *back-end* berisi tabel-tabel yang berkaitan dengan proses *back-end* yang ada pada *database server*. Fragmentasi lainnya berisi tabel tabel yang berkaitan dengan proses proses registrasi dan fragmentasi yang berisi tabel tabel yang berkaitan dengan proses transaksi. Untuk menampung transaksi ada dua *database* yang pemetaannya menggunakan replikasi yaitu data yang sama dengan data mainnya (data transaksi1). *Database* replikasi transaksi ini dihubungkan dengan dengan *server* transaksi 1 yang menangani proses registrasi, agar saat terjadi pengalihan transaksi data dari *server* transaksi 2 ke transaksi 1 data bisa dikirim ke *database* replikasi transaksi ini, yang kemudian akan dikirimkan ke *database* transaksinya utamanya. Hal ini dilakukan karena pada registrasi tidak terlalu menerima banyak beban komputasi sedangkan transaksi menerima banyak beban sehingga *server* transaksi1 menangani dua *database* sehingga lebih efisien. Pentransaksian data dari masing masing *database* tidak dilakukan dengan cara asinkron dimana setiap pengiriman data dilakukan secara terus menerus ke *data base* pusat, karena hal ini tidak begitu membantu dalam mengurangi lalulintas data. Pentransaksian data dalam model ini lebih cocok menggunakan metode sinkron dimana setiap data dikumpulkan terlebih dahulu baru setelah ada pemicu data akan dikirimkan secara bersama, dalam hal ini pemicu yang cocok adalah kesibukan *server*. Jadi jika *server* yang dituju sudah tidak terlalu sibuk, baru *database* antrian yang akan mengirimkan datanya mulai melakukan pengiriman data.

Jadi dengan adanya pemisahan secara fisik seperti ini beban komputasi bisa tersebar dan tidak terpusat ke suatu tempat. Untuk melakukan distribusi ini bisa dilakukan dengan menggunakan RMI dari JAVA atau Webservice dari C# maupun PHP. Hanya saja untuk mengembangkan sistem terdistribusi ini terdapat hambatan yaitu pada masalah jaringan, dimana memerlukan biaya yang mahal untuk proses penginstalan jaringan yang memiliki kecepatan transafer tinggi.

4. KESIMPULAN

1. Untuk membangun arsitektur sistem terdistribusi dalam suatu perusahaan sangatlah dipengaruhi oleh proses bisnis perusahaan.
2. Masalah terkini yang dihadapi dalam pembangunan sistem terdistribusi adalah kecepatan jaringan yang ada pada saat ini.
3. Pemisahaan suatu sistem digunakan untuk pembagian komputasi sehingga komputasi yang terjadi tidak terpusat, yang dapat mengakibatkan *bottleneck effect*.
4. Banyaknya sistem yang didistribusikan sangatlah tergantung dengan arus data yang mengakibatkan terjadinya komputasi tinggi dan juga kebutuhan perusahaan.
5. Pada kasus tertentu penggunaan *nested transaction* tidak cukup sehingga dibutuhkan flat *transaction*.

DAFTAR PUSTAKA

[6] Abdouli M. , Sadeg B. , Amanton L. , *Scheduling Distributed Real-Time Nested Transactions,* Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05), ISBN: 0-7695-2356-0

[4] Agrawal Kunal, Leiserson Charles E., Sukha Jim*, 2002, Memory Models for Open Nested Transactions*

[5] Chen, Hong-Ren , Y. H. Chin***, Scheduling Value-Based Nested Transactions in Distributed Real-Time Database Systems*, 2004,** Real-Time Systems, Volume 27, Pages: 237 - 269

[2] Eliot, J., B. Moss, 2004, *Open Nested Transactions: Semantics and Support*

[3] Gançarski Stephane, León Claudia, Naacke Hubert, Rukoz Marta, Santini Pablo, 2006, *Integrity Constraint Checking in Distributed Nested Transactions over a Database Cluster*, Clei Electronic Journal, Volume 9

[1] Herder, Theo, Rothermel Kurt,2007, *Concurrency Control Issues in Nested Transactions*

[7] KAHFI , MUHAMMAD RIJALUL, IBNU HANIF, *MAKALAH DISTRIBUTED TRANSACTION (*http://te.ugm.ac.id/~risanuri/v01/wp-content/uploads/2009/06/distributed_*transaction*.pdf)

[8] INDRAJIT, RICHARDUS EKO, EVOLUSI STRATEGI INTEGRASI SISTEM INFORMASI RAGAM INSTITUSI, KIAT MEMECAHKAN PERMASALAHAN POLITIS DALAM KERANGKA MANAJEMEN PERUBAHAN SEKOLAH TINGGI MANAJEMEN INFORMATIKA DAN KOMPUTER STIMIK PERBANAS , JAKARTA, INDONESIA