Rancang Bangun *Fault Tolerance* pada Sistem *Database* untuk Aplikasi *Point Of Sale*

S. Oei dan A. Ashari

Abstract— *A fault tolerant system is a system that has the ability to be able to continue its jobs properly although there is a fault in hardware or software of the system. A system that has a fault tolerant capability is usually critical or important system. For example is at Point Of Sale application. Point Of Sale Application has evolved from offline-based toward online-based. With an online-based, the existence of a central database server must be guaranteed free from failure.*

To get a central database server that is free from failure is impossible. Something that can be cultivated is to create a system that can tolerate a failure in the central database server. This is done with the help of a second central database server (slave), which is useful as a replication of the central database server (master). To ensure data from these two central database servers are the same, then used the concept of reading and writing "read one/write all". And to manage all the processes needed in implementing a fault tolerant application, we need the help from a coordinator.

The result obtained in this research is a design of fault tolerant architecture that can be applied to various types of information system applications include Point Of Sale application. By using this fault tolerant architecture that has been built, Point Of Sale application can still run the transaction process even if there is a failure in accessing data in one central database server (master/slave).

Keywords— *fault tolerance, database system, point of sale application*

1. PENDAHULUAN

Dewasa ini penggunaan sistem komputer telah meluas ke berbagai aspek kehidupan manusia. Baik untuk sistem pertahanan, sistem penerbangan, sistem kendali lalu-lintas, sistem perbankan, ataupun untuk sistem informasi. Berbagai aplikasi telah dibuat dengan menggunakan sistem komputer guna menunjang aktivitas manusia. Hal ini mengakibatkan adanya ketergantungan manusia akan sistem komputer.

Standy Oei, Mahasiswa Program Pascasarjana Ilmu Komputer, Fakultas Matematika & Ilmu Pengetahuan Alam, UGM, Yogyakarta

Ahmad Ashari, Staf Pengajar Program Pascasarjana Ilmu Komputer, Fakultas Matematika & Ilmu Pengetahuan Alam, UGM, Yogyakarta.

Ketika sistem komputer mengalami masalah, maka aktivitas yang dikerjakan manusia itu akan terhambat ataupun terhenti sama sekali. Dimulai dari sinilah muncul suatu pemikiran untuk membuat suatu sistem komputer yang memiliki kemampuan toleransi terhadap masalah kegagalan sistem yang bisa akan muncul (*fault tolerant system*). Dimana salah satu masalah kegagalan sistem yang bisa muncul yakni pada sistem *database*. Dan hampir semua aplikasi sistem informasi dibuat menggunakan *database* sebagai tempat penyimpanan data.

Aplikasi *Point Of Sale* (POS) merupakan salah satu contoh dari aplikasi sistem informasi, yang digunakan untuk mempermudah proses pencatatan transaksi penjualan barang, dimana aplikasi ini biasa ditempatkan di lokasi pembayaran barang (kasir) seperti yang ada di minimarket ataupun supermarket.

Aplikasi *Point Of Sale* telah berkembang dari berbasis *offline* menuju ke berbasis *online*. Dengan berbasis *online*, semua aplikasi *Point Of Sale* yang tersebar di semua kasir penjualan akan sangat bergantung terhadap eksistensi dari *server database* pusat. Dari sinilah muncul adanya kebutuhan untuk mengaplikasikan kemampuan *fault tolerant* ke dalam aplikasi *Point Of Sale* agar bisa mentolerir kegagalan yang mungkin saja akan terjadi pada *server database* pusat.

Masalah *fault tolerant*, aplikasi *Point Of Sale*, beserta *query processing* telah banyak dikembangkan dalam berbagai penelitian. Beberapa penelitian yang telah membahas masalah berkaitan dengan penelitian ini diantaranya adalah sebagai berikut.

Pada penelitian [1] menunjukkan bukti bahwa pengembangan sebuah aplikasi menggunakan *query processing* bisa dilakukan dan bahkan membuat lebih mudah bagi *programmer* dalam membangun aplikasi. *Software* DBMS yang digunakan adalah MySQL, dimana MySQL ini merupakan *software* DBMS yang *open source* dan *powerful*. Selain itu juga, MySQL membangun sistemnya berdasarkan *query processing*.

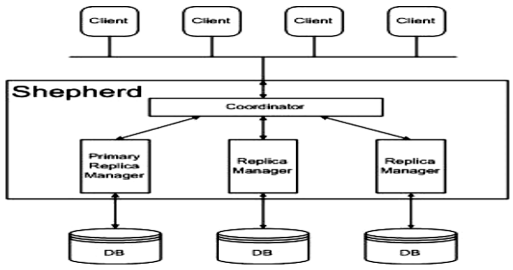
Pada penelitian [2] membahas tentang perancangan dan pembuatan Sistem Informasi

Akademik yang *Fault Tolerance* menggunakan metode replikasi data antar *database*. Dimana dalam mensikronkan data atau *recovery* data antar *database* digunakan cara *backup* dan *restore* data dari satu *database* ke *database* yang lain.

Pada penelitian [3] membahas *software fault tolerant* yang diaplikasikan pada sistem terdistribusi. *Fault tolerant* ini diaplikasikan pada *fieldbus* yang dipergunakan dalam sistem terdistribusi. Saat ini sudah banyak *fieldbus* yang dipergunakan di industri, di antaranya *Control Area Network* (CAN), Profibus, I2C, dan lain-lain. Dalam pembahasan ini menggunakan CAN dan SPORT (*Serial Port*) untuk membangun sistem terdistribusi dengan *fault tolerant*.

Pada penelitian [4] membahas tentang teknik yang digunakan dalam meningkatkan *service availability*, seperti melalui teknik *redundancy*, *machine virtualization*, dan *autonomic service placement*. Konsep *availability* didefinisikan sebagai kemungkinan/probabilitas sebuah sistem untuk bisa hidup dan siap untuk digunakan. *Availability* bisa dirumuskan lewat sebuah persamaan berikut ini: *availability = MTBF / (MTBF + MTTR)*, dimana MTBF = *mean time between failure* dan MTTR = *mean time to repair*.

Pada penelitian [5] membuat suatu rancangan *Heterogeneous Replicated Database* (HRDB). HRDB merupakan suatu rancangan *fault tolerant* DBMS yang mengimplementasikan sebuah skema replikasi praktis dalam mengatasi permasalahan kegagalan database (*database faults*). Seperti pada penelitian yang telah ada sebelumnya, HRDB menggunakan teknik replikasi, tetapi HRDB menggabungkan dua ide kunci dalam mengatasi permasalahan yang telah muncul sekarang ini. Pertama, untuk mencapai *failure-independence*, HRDB menggunakan *heteregeneous replicator*. Kedua, untuk mengatasi *Byzantine failures*, HRDB menggunakan *voting*. Arsitektur rancangan HRDB yang dibuat adalah seperti pada Gambar 1.



Gambar 1 Arsitektur perancangan HRDB

2. METODE PENELITIAN

2.1 Analisa Sistem

Sistem yang akan dirancang dan dibangun dalam penelitian ini adalah sistem yang memiliki kemampuan *fault tolerant* dalam mengatasi permasalahan kegagalan pengaksesan data pada salah satu *server database* pusat, yang diakibatkan oleh kerusakan komponen. Kerusakan yang dimaksud seperti putusnya koneksi kabel jaringan pada *server database* pusat, serta kerusakan secara permanen pada komponen penting di *server database* pusat yang menunjang ketersediaan data (seperti prosesor, hardisk, memori, kartu jaringan (*network card*), *motherboard* dan *power supply*).

Contoh penerapan yang diangkat disini adalah pada aplikasi *Point Of Sale*. *Server database* pusat yang akan digunakan berjumlah 2 buah, yakni *master* dan *slave*. Dalam mengakses data pada *server database* pusat, *client* tidak berinterakasi langsung dengan *server database* pusat. Melainkan *client* harus berkomunikasi dengan sebuah *coordinator*. Dimana dalam *coordinator* inilah fungsi *fault tolerant* itu dilakukan.

Coordinator berfungsi sebagai pengeksekusi *request* (*query*) dari *client* dan sekaligus berguna sebagai *buffer* untuk menyimpan *log* dari *query* yang gagal untuk dieksekusi pada *server database* pusat (*master* ataupun *slave*). *Coordinator* bisa secara pintar memilih menggunakan *server database* pusat mana yang sedang hidup. Dan setelah *server database* pusat (*master* atau *slave*) yang gagal diakses sebelumnya bisa kembali bekerja normal, maka isi dari *buffer* (pada *coordinator*) yang berisi *query* bisa dikosongkan dengan cara dengan mengeksekusi kembali semua *query* yang gagal dieksekusi sebelumnya (proses *recovery*).

Aplikasi *Point Of Sale* yang akan dibangun menggunakan metode *one-copy serializability* [6] dalam proses pengaksesan data pada kedua *server database* pusat baik *master* ataupun *slave*. Implementasi dari metode ini bisa diilustrasikan dengan kata *read one/write all*, yakni aplikasi *Point Of Sale* akan melakukan pembacaan data dari salah satu *server database* pusat dan akan melakukan penulisan ke semua *server database* pusat yang ada. Hal ini berguna untuk menjamin isi dari kedua *server database* pusat tersebut mempunyai isi data yang sama.

Dengan adanya kemampuan *fault tolerant* ini, aplikasi *Point Of Sale* bisa terus melakukan

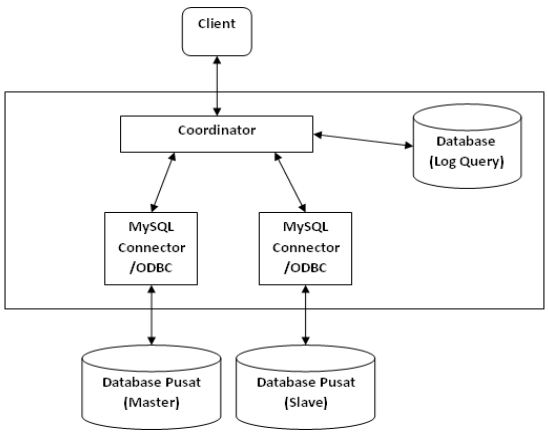
kerjanya walaupun terdapat salah satu *server database* pusat yang mati.

Sistem yang akan dibangun ini mempunyai karakteristik sebagai berikut:

1. Aplikasi *Point Of Sale* yang akan dibangun merupakan aplikasi yang berbasis *desktop*.
2. Aplikasi *Point Of Sale* dijalankan pada sistem jaringan LAN (*Local Area Network*).
3. Aplikasi *Point Of Sale* (*Client*), *Coordinator*, dan *Server Database* pusat dijalankan pada sistem operasi *windows*.
4. *Coordinator* diasumsikan akan selalu hidup atau bisa diakses agar fungsi *fault tolerant* yang telah dibangun bisa berjalan.
5. Masalah utama yang diperhatikan adalah kemampuan dari aplikasi *Point Of Sale* dalam mentolerir kegagalan pengaksesan dari salah satu *server database* pusat, dan melakukan proses *recovery* data dengan menggunakan *log query* yang ada.
6. Interkoneksi antara *client*, *coordinator*, dan *server database* pusat masih sepenuhnya mengandalkan kemampuan dari Winsock dan MySQL Connector atau ODBC.

2.2.Perancangan Arsitektur Sistem Fault Tolerant

Dalam membangun suatu sistem yang *fault tolerant* diperlukan adanya suatu analisa terhadap perancangan dari sistem tersebut. Analisa diperlukan untuk bisa menggambarkan suatu arsitektur yang menunjang penerapan *fault tolerance* di dalam sistem. Suatu arsitektur yang dibangun haruslah bisa menjelaskan bagaimana suatu proses *fault tolerant* dilakukan di dalam sistem. Gambar 2 merupakan rancangan dari arsitektur *fault tolerant* yang akan dibangun.



Gambar 2 Rancangan arsitektur *fault tolerant*

Pada rancangan ini, tidak dibutuhkan adanya modifikasi terhadap *software database* yang digunakan. Dan tidak diperlukan adanya suatu *software* khusus yang dijalankan pada *server database* pusat (*master* ataupun *slave*). Hal ini didukung karena *software* DBMS yang digunakan sudah menawarkan suatu standar antarmuka SQL dan menyediakan protokol komunikasi yakni ODBC.

Client tidak berinteraksi secara langsung dengan setiap *server database* pusat (*master* atau *slave*). Tetapi *client* berkomunikasi dengan sebuah *coordinator*, yang bertindak sebagai sebuah *front-end* ke *database* dan mengatur atau mengkoordinasikan *client* yang ada.

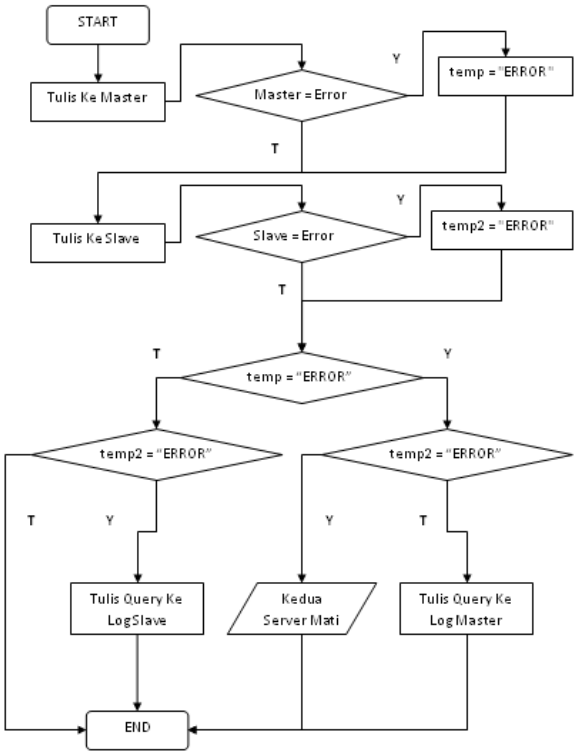
Untuk melakukan penulisan (*write*), *client* mengirimkan SQL *statement* ke *coordinator*. SQL *statement* yang tiba di *coordinator* kemudian akan dieksekusi ke kedua *server database* pusat (*master* dan *slave*) lewat protokol komunikasi ODBC. Dimana setiap *server database* pusat memiliki koneksi ODBC masing-masing. Ketika terdapat suatu *server database* pusat (*master* atau *slave*) yang gagal mengeksekusi SQL *statement* yang telah diberikan tersebut, maka *coordinator* akan mencatat SQL *statement* (*query*) yang gagal dieksekusi ke dalam *database* (*log query*). Ketika sukses mengeksekusi SQL *statement* yang dikirim oleh *client*, *coordinator* akan mengirim kembali respon atau jawaban kepada *client*. Dalam berkomunikasi antara *client* dan *coordinator* digunakan protokol komunikasi Winsock.

Ketika *server database* pusat yang telah mati sebelumnya bisa kembali beroperasi, maka semua catatan *query* yang gagal dieksekusi ke dalam *server database* pusat tersebut bisa dieksekusi kembali. Sekaligus isi dari *database* (*log query*) bisa kembali dikosongkan ketika *query* yang gagal dieksekusi tersebut bisa sukses dieksekusi kembali. Dimana semua proses *recovery* ini dikendalikan atau dilakukan sepenuhnya oleh *coordinator*.

Untuk melakukan pembacaan (*read*), *client* melakukan pemanggilan suatu prosedur yang ada di dalam *coordinator*. Hal ini dilakukan dengan cara mengirimkan nama prosedur dan informasi tambahan yang dibutuhkan dalam menjalankan prosedur tersebut ke *coordinator*. Kemudian *coordinator* akan menjalankan prosedur pembacaan tersebut ke salah satu *server database* pusat (*master* atau *slave*) yang masih hidup. Setelah mendapatkan hasil pembacaan, maka *coordinator* akan mengirimkannya ke *client*.

Pada intinya, proses penulisan data dilakukan pada kedua *server* (*write all*) dan proses pembacaan data dilakukan pada salah satu *server*

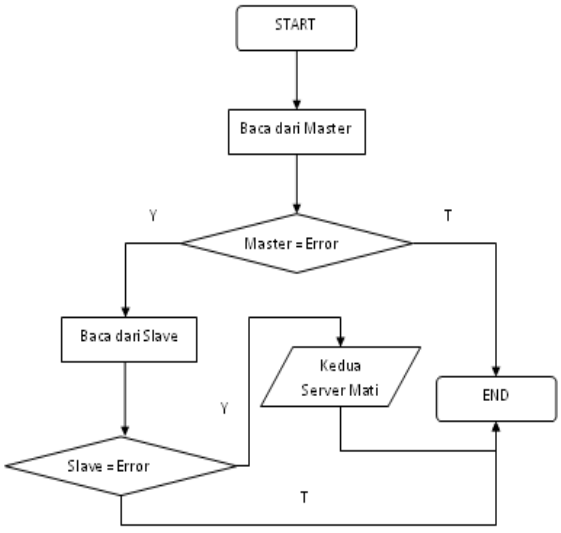
(*read one*). Hal ini dilakukan untuk menjamin kedua *server database* pusat mempunyai isi data yang sama. Gambar 3 memperlihatkan proses penulisan data pada kedua *server database* pusat beserta cara penanggulangan jika ada *server database* pusat yang mengalami kegagalan atau mati.



Gambar 3 *Flowchart* proses penulisan data

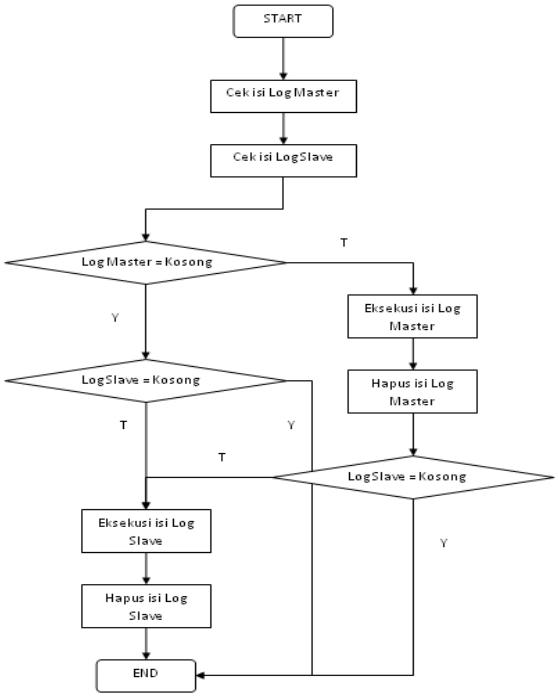
Proses tulis ke *Master* atau *Slave* terdiri dari proses *insert*, *edit* dan *delete*.

Pada Gambar 4 memperlihatkan proses pembacaan dari salah satu *server database* pusat yang masih hidup.



Gambar 4 *Flowchart* proses pembacaan data

Server database pusat yang mati tentunya mempunyai catatan *query* yang belum atau gagal dieksekusi di dalam *database* (*log query*). Dan sebelum *server database* pusat tersebut bisa kembali beroperasi haruslah dilakukan proses sinkronisasi terlebih dahulu. Untuk itu, semua *query* yang gagal dieksekusi sebelumnya haruslah bisa dieksekusi kembali. *Query* yang telah berhasil dieksekusi kembali bisa dihapus dari dalam *database* (*log query*). Gambar 5 memperlihatkan proses *recovery* data pada *server database* pusat.



Gambar 5 *Flowchart* proses *recovery* data

2.3 Implementasi Sistem

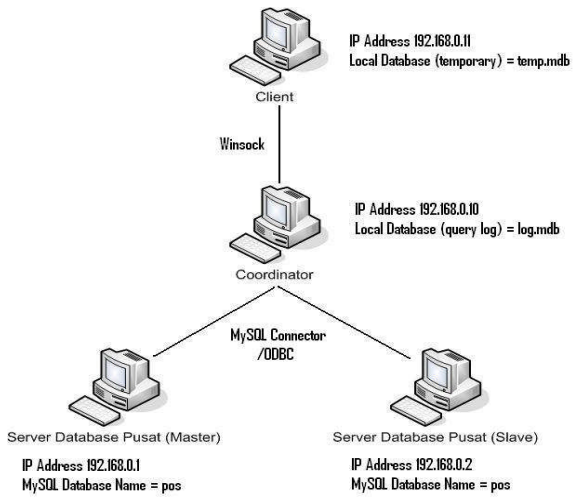
Pada penelitian ini, pembuatan aplikasi *Point Of Sale* yang *fault tolerant* diimplementasikan dengan menggunakan bahasa pemrograman Visual Basic 6.0. Implementasi sistem ini sesuai dengan arsitektur yang telah dirancang sebelumnya. Dalam implementasi, sistem ini menggunakan peralatan sebagai berikut:

1. Perangkat keras
   a. AMD Turion X2 Dual-Core Mobile RM-74 2,2 GHz
   b. Hardisk 250 GB
   c. Memori RAM 4 GB - Dual Channel (128-bit)
2. Perangkat lunak
   a. Microsoft Windows XP
   b. XAMPP for Windows 1.7.2 (MySQL 5.1.37)

c. SQLyog Community Edition- MySQL GUI 8.12
d. MySQL Connector/ODBC 5.1
e. Microsoft Access 2003
f. Visual Basic 6.0
g. Oracle VirtualBox 3.2.6 r63112

Simulasi jaringan untuk sistem yang dibuat menggunakan bantuan *software* Virtual Machine yakni Virtual Box. Untuk *client*, *coordinator*, dan *server database* pusat menggunakan sistem operasi Windows XP. *Server database* pusat menggunakan *software* DBMS MySQL. Dan untuk *database* (*log query*) beserta *database temporary* pada *client* menggunakan *software* DBMS Microsoft Access 2003.

Sebelum bisa mengimplementasikan pembuatan aplikasi *Point Of Sale* yang *fault tolerant*, kita haruslah terlebih dahulu membangun arsitektur jaringan yang akan menunjang pengimplementasian aplikasi tersebut. Pada Gambar 6 akan memperlihatkan rancangan arsitektur jaringan yang digunakan dalam membangun sistem.



Gambar 6 Arsitektur jaringan sistem yang dibangun

Setelah memiliki jaringan yang bisa digunakan, barulah kita bisa segera mengimplementasikan pembuatan aplikasi yang kita inginkan. Agar penerapan arsitektur *fault tolerant* ini bisa berjalan dengan baik, maka *availability* dari *coordinator* haruslah bisa dijaga dengan baik.

Pengimplementasian dari aplikasi *Point Of Sale* yang *fault tolerant* meliputi:

- Proses Pembacaan (*Read One*)
- Proses Penulisan (*Write All*)
- Proses *Recovery*
  - Pada awal mulai dari *coordinator*
  - Pada setiap selang waktu tertentu

3. HASIL DAN PEMBAHASAN

3.1. Hasil Percobaan

Sistem yang telah dirancang dan dibangun ini telah memperlihatkan kemampuannya dalam mengatasi kegagalan pengaksesan data pada salah satu *server database* pusat. Aplikasi *Point Of Sale* tersebut bisa tetap menjalankan tugasnya walaupun terdapat salah satu *server database* pusat yang mati. Dari percobaan yang dilakukan memperlihatkan sistem telah bisa menerapkan konsep pembacaan pada salah satu *server database* pusat dan penulisan terhadap semua *server database* pusat yang ada. Selain itu, sistem mempunyai kemampuan *recovery* untuk melakukan proses sinkronisasi data.

Percobaan yang dilakukan dengan mencoba melakukan pembacaan dan penulisan ketika *server database* pusat hidup keduanya, hanya *server database* pusat (*master*) yang hidup, hanya *server database* pusat (*slave*) yang hidup, ataupun keduanya mati.

3.2. Pembahasan

3.2.1. Percobaan dengan master dan slave hidup

Percobaan yang pertama dilakukan adalah dengan keadaan dimana kedua *server database* pusat hidup atau terkoneksi dengan baik. Dimana untuk operasi pembacaan, *coordinator* hanya akan melakukan pembacaan dari *server database* pusat (*master*). Dan untuk operasi penulisan akan dilakukan pada kedua *server database* pusat.

3.2.2. Percobaan dengan master hidup dan slave mati

Percobaan yang kedua dilakukan adalah dengan keadaan dimana *server database* pusat (*master*) hidup atau terkoneksi dengan baik, sedangkan *server database* pusat (*slave*) mati atau tidak terkoneksi. Dimana untuk operasi pembacaan, *coordinator* hanya akan melakukan pembacaan dari *server database* pusat (*master*). Dan untuk operasi penulisan akan hanya dilakukan pada *server database* pusat (*master*). Untuk keperluan proses *recovery* nantinya, operasi penulisan juga dilakukan terhadap tabel *server2* yang berada pada *database* (*log query*). Hal ini berguna untuk mencatat *query* penulisan yang gagal dieksekusi pada *server database* pusat (*slave*).

3.2.3. Percobaan dengan master mati dan slave hidup

Percobaan yang ketiga dilakukan adalah dengan keadaan dimana *server database* pusat (*master*) mati atau tidak terkoneksi, sedangkan

server database pusat (*slave*) hidup atau terkoneksi dengan baik. Dimana untuk operasi pembacaan, *coordinator* hanya akan melakukan pembacaan dari *server database* pusat (*slave*). Dan untuk operasi penulisan akan hanya dilakukan pada *server database* pusat (*slave*). Untuk keperluan proses *recovery* nantinya, operasi penulisan juga dilakukan terhadap tabel *server1* yang berada pada *database* (*log query*). Hal ini berguna untuk mencatat *query* penulisan yang gagal dieksekusi pada *server database* pusat (*master*).

3.2.4. Percobaan dengan master dan slave mati

Percobaan yang keempat dilakukan adalah dengan keadaan dimana kedua *server database* pusat (*master* dan *slave*) mati atau tidak terkoneksi. Pada keadaan seperti ini, *coordinator* akan berhenti melakukan tugasnya atau keluar. Hal ini dikarenakan untuk melakukan transaksi, aplikasi *Point Of Sale* perlu untuk melakukan pembacaan data barang seperti harga barang dari dalam *server database* pusat.

4. KESIMPULAN

Berdasarkan penelitian yang telah dilakukan, maka diambil beberapa kesimpulan sebagai berikut:

1. Penerapan arsitektur *fault tolerant* bisa dilakukan pada aplikasi *Point Of Sale*.
2. Arsitektur *fault tolerant* yang telah dibangun ini memiliki kemampuan dalam mengatasi permasalahan *fail-silent fault* pada salah satu *server database* pusat (*master/slave*).
3. Aplikasi *Point Of Sale* tetap bisa menjalankan proses transaksi walaupun terdapat salah satu *server database* pusat yang mati. Hal ini dikarenakan *coordinator* akan secara pintar memilih menggunakan *server database* pusat yang masih hidup.
4. Di bawah kendali dari *coordinator*, bila terdapat *query* penulisan yang gagal dieksekusi pada *server database* pusat (*master* atau *slave*), maka *query* tersebut akan dicatat pada *database* (*log query*) yang nantinya akan digunakan untuk proses *recovery*.
5. Proses *recovery* dilakukan pada 2 kondisi, yakni pada awal mulai dari *coordinator* atau pada setiap selang waktu tertentu.

DAFTAR PUSTAKA

[1] Budiyanto, A., 2005, Rancang Bangun Aplikasi Point Of Sale Berbasis Query Processing, *Tesis*, Magister Ilmu Komputer FMIPA UGM, Yogyakarta.

[2] Setyorini, T. A., 2010, Rancang Bangun Untuk Sistem Informasi Akademik yang Fault Tolerance, *Tesis*, Magister Ilmu Komputer FMIPA UGM, Yogyakarta.

[3] Ferdinando, H., 2004, Membangun Sistem Fault Tolerant Pada Fieldbus Untuk Aplikasi Sistem Terdistribusi, *Seminar Nasional Ilmu Komputer dan Teknologi Informasi (SNIKTI)*, Bogor, 2-3 September 2004.

[4] Boon, Y. O., Huah, Y. C., dan Yu-N, C., 2010, Dynamic Service Placement and Redundancy: Concept and Research Direction, *Proceedings of The Second International Conference on Distributed Framework & Applications 2010*, hlm. 47-52, Yogyakarta, 2-3 Agustus 2010.

[5] Vandiver, B., Balakrishnan, H., Liskov, B., dan Madden, S., 2006, Database Fault-Tolerance with Heterogeneous Replication, *http://publications.csail.mit.edu/abstracts/abstracts06/benmv/benmv.html*, diakses tanggal 18 Januari 2011.

[6] Coulouris, G., Dollimore, J., dan Kindberg, T., 2000, *Distributed Systems Concepts and Design*, 3rd ed., Pearson Education Limited, China.