**IJCCS**, Vol.6, No.2, July 2012, pp. 89~100
ISSN: 1978-1520


# Implementasi Basisdata Terdistribusi Pada Sistem Kenaikan Gaji Berkala Dinas Infokom Provinsi Maluku

**Anastasya Latubessy*[1], Ahmad Ashari [2]**
[1]Program Studi S2 Ilkom, FMIPA, UGM, Yogyakarta
[2]Jurusan Ilmu Komputer dan Elektronika, FMIPA UGM, Yogyakarta
e-mail: ***[1]anastasyalatubessy@ugm.ac.id** , [2]ashari@ugm.ac.id

***Abstrak***

*Kemampuan socket yang bekerja pada komunikasi low level dalam melakukan transfer data antar aplikasi, diharapkan dapat diterapkan dalam sistem basis data terdistribusi. Beberapa alasan untuk membangun basis data terdistribusi adalah pemakaian bersama, kehandalan, ketersediaan, dan kecepatan pemrosesan query. Dengan pertimbangan tersebut, instansi-instansi baik pemerintah maupun swasta mulai tertarik untuk menggunakan sistem terdistribusi dalam pengelolaan pekerjaan pada instansinya. Hal ini dijadikan alasan untuk membangun sistem kenaikan gaji berkala otomatis di Dinas Informasi dan Komunikasi (Infokom) Provinsi Maluku. Selain itu, belum adanya sistem terkomputerisasi yang menangani sistem ini. Sistem kenaikan gaji berkala merupakan sistem yang bersifat kritis atau penting karena berhubungan dengan masalah uang, sehingga diperlukan juga sebuah sistem yang fault tolerant.*

*Arsitektur sistem dibangun dengan metode client server yang memanfaatkan socket sebagai middleware dalam proses komunikasi pada jaringan. Menggunakan pendekatan full replica dengan pendekatan file replication using a group untuk menjamin konsistensi data. SQL Statement yang dikirimkan client akan diubah kedalam format XML yang kemudian akan dikirimkan ke kedua basisdata server menggunakan socket. Model redundansi data dan rendundansi proses diatur oleh middleware socket.*

*Hasil yang diperoleh dalam penelitian ini adalah aplikasi socket sebagai middleware yang menangani distribusi data dari aplikasi web client ke basisdata server yang diimplementasikan pada sistem kenaikan gaji berkala otomatis di Dinas Infokom Provinsi Maluku.*

***Kata kunci****— Socket, basisdata terdistribusi, client server*

***Abstract***

*Sockets ability to transfer data between application that work on the low level communication was expected to be implemented in a distributed database system. Some of the reasons for building a distributed database was data sharing, reliability, availability, and query processing speed. With these considerations, both government agencies and private sector became interested to use distributed systems to manage their job at the office. It is used as a reason to build an automated KGB system of Infokom Department in Province of Moluccas. Furthermore, there was no computerized system to handle this system before. Because of it was a critical or important system, then it was requires to build a system that has a fault tolerant capability.*

*The system architecture was built by client server method using sockets as middleware in the process of network communicaton. The system method was used full replica, and used file replication using a group to ensure data consistency. The SQL Statement was sent by the client would be converted into an XML format which then would be sent to the server using a socket. Data redundancy model and process redundancy model managed by socket middleware.*

*Received April 5th,2012; Revised June 20th, 2012; Accepted June 29th, 2012*

*The results of this research was a middleware application using socket that handle the distribution of databases from client applications to database server that were implemented on a periodic system of automatic salary increases in the Office of Information and Communication in Province of Moluccas.*

***Keywords****— Socket, distributed database, replication*

## 1. PENDAHULUAN

Sebuah sistem basisdata terdistribusi terdiri dari kumpulan *site-site*, masing-masing *site* ini dapat berpartisipasi dalam pemrosesan transaksi yang mengakses data pada suatu *site* atau beberapa *site*. Beberapa alasan untuk membangun basisdata terdistribusi, seperti pemakaian bersama (*share*), kehandalan (*reliability*), ketersediaan (*availability*) dan kecepatan pemrosesan *query*[1]. Sedangkan socket merupakan jembatan yang memungkinkan terjadinya komunikasi dalam jaringan komputer, dengan kata lain bisa disebut *middleware*[2].

Berdasarkan tinjauan tersebut, pada penelitian ini dibangun aplikasi *middleware* socket sebagai jembatan yang menghubungkan komunikasi antara aplikasi *client* dan dua basisdata *server* yang dibangun. *Middleware* socket digunakan untuk menangani proses replikasi data dari *client* ke kedua basisdata *server* yang terhubung (basisdata *master* dan basisdata *slave*). *Middleware* socket juga menangani masalah kegagalan akses data pada basisdata *server master*, dan melakukan proses *recovery* data saat basisdata *server master* hidup kembali.

Terdapat dua strategi utama dalam proses desain basisdata distribusi yaitu pendekatan *top-down* dan pendekatan *bottom-up.* Pendekatan *top-down* digunakan jika melakukan perancangan dari awal sistem dikembangkan. Biasanya dalam sistem yang homogen. Sedangkan pendekatan *bottom up* digunakan apabila basisdata sudah ada pada beberapa *server*. Pada pendekatan ini basisdata harus dihubungkan untuk menyelesaikan tugas[3]. Pada penelitian ini menggunakan pendekatan *top-down*. Karena sistem dibangun dari awal dan belum ada basisdata pada *server*.

Alasan utama membangun sistem kenaikan gaji berkala otomatis di Dinas Informasi dan Komunikasi (Infokom) Provinsi Maluku adalah belum adanya sistem terkomputerisasi yang menangani sistem ini. Sistem kenaikan gaji berkala merupakan sistem yang bersifat kritis atau penting karena berhubungan dengan masalah uang, sehingga diperlukan sebuah sistem yang *fault tolerant*. Sistem yang *fault tolerant* merupakan suatu sistem yang memiliki kemampuan untuk dapat melanjutkan tugasnya dengan benar walaupun terjadi *fault* pada *hardware* ataupun *software* dari sistem tersebut. Suatu sistem yang memiliki kemampuan *fault tolerant* biasanya merupakan sistem yang bersifat kritis atau penting (*critical system*). Hal ini biasanya dihubungkan dengan masalah uang dan keselamatan[4].

Berdasarkan hal tersebut, maka dirancang sebuah aplikasi sistem informasi kepegawaian yang akan menangani proses pengumpulan data pegawai untuk kenaikan gaji berkala di Dinas Infokom Provinsi Maluku berbasis *client server* dengan dua *server* basisdata yang ditempatkan pada bagian keuangan dan bagian perencanaan. Sementara aplikasi web sistem informasi yang dibangun ditempatkan di bagian kepegawaian.

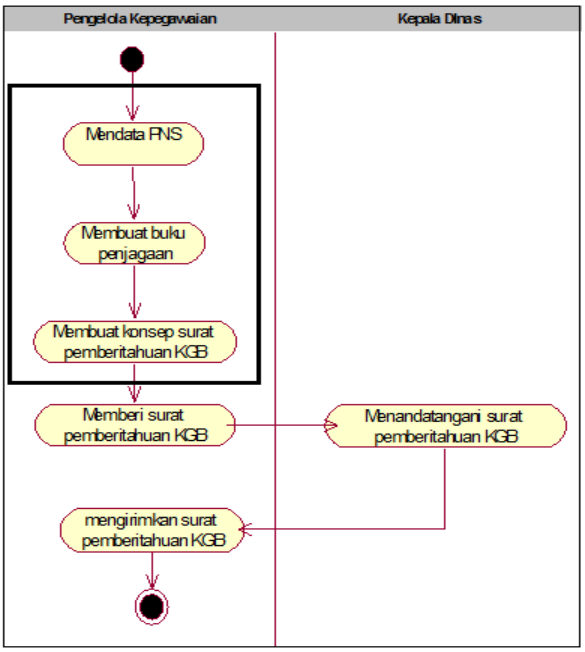
## 2. METODE PENELITIAN

### *2.1 Analisa*

Proses bisnis didefenisikan sebagai aktivitas yang terukur dan terstruktur untuk memproduksi output tertentu untuk kalangan pelanggan tertentu[5]. Prosedur pembuatan surat pemberitahuan kenaikan gaji berkala bagi pegawai negeri sipil antara lain:

a. Pengelola kepegawaian mendata pegawai negeri sipil yang telah memenuhi persyaratan untuk kenaikan gaji berkala.
b. Pengelola kepegawaian membuat penjagaan kenaikan gaji berkala.
c. Pengelola kepegawaian membuat konsep surat pemberitahuan kenaikan gaji berkala untuk ditandatangani kepala biro.
d. Setelah surat pemberitahuan kenaikan gaji berkala ditandatangani oleh kepala biro, pengelola kepegawaian mengirimkan surat pemberitahuan kenaikan gaji berkala kepada Badan Pengelolaan Keuangan dan Kekayaan Negara beserta tembusan-tembusannya.

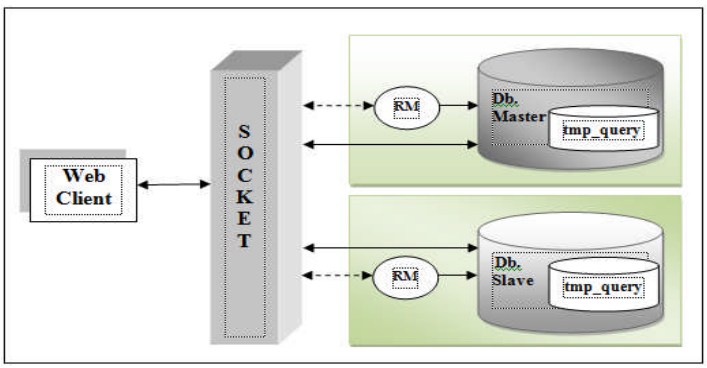
Gambar 1 menunjukkan aliran akitivitas yang terjadi dalam pembuatan KGB di Dinas Infokom Provinsi Maluku. Bagian yang berada didalam kotak merupakan bagian yang dibuatkan kedalam perangkat lunak.



Gambar 1 Aliran bisnis proses KGB

*2.2 Perancangan Sistem*

Rancangan sistem *middleware* socket yang dibangun juga mendukung adanya *fault tolerant.* Sehingga rancangan *fault tolerant* yang dibangun dapat dilihat pada Gambar 2. Proses replikasi data dari *client* ke kedua *server* basisdata dilakukan melalui *middleware* socket. Tabel *query* pada *server* basisdata digunakan saat *recovery*.



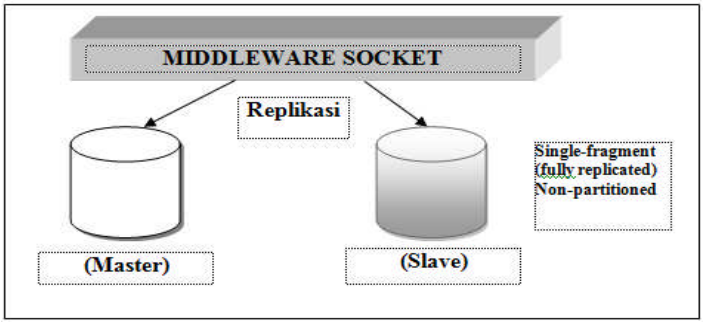
Gambar 2 Rancangan arsitektur sistem dengan *middleware* socket

*Middleware* socket mengecek koneksi ke *master* dan *slave* dengan menjalankan *trigger*. *Trigger* tersebut berisi proses perulangan dimana proses tersebut akan mengecek koneksi ke *master* dan *slave* benar-benar tersambung. Jika koneksi pada basis data *master* tidak tersambung maka koneksi *client* yang awalnya ke *master* dipindahkan ke *slave*. Kemudian *trigger* juga menjalankan proses untuk menyimpan *query* ke *temporary query* di *slave*. Saat proses di *trigger* mendeteksi bahwa koneksi ke *master* telah tersambung maka *trigger* akan menjalankan proses *recovery*. Namun, jika *trigger* mendeteksi koneksi pada basis data *slave* tidak tersambung maka jika saat itu koneksi sedang berada pada basis data *slave*, koneksi akan dipindahkan ke basis data *master*. Kemudian *trigger* juga menjalankan proses untuk menyimpan *query* ke *temporary query* di *master*. Saat proses di *trigger* mendeteksi bahwa koneksi ke *slave* telah tersambung maka *trigger* akan menjalankan proses *recovery*.

Proses *recovery* pada basis data *master* adalah *middleware* mengambil data dari tabel *temporary query* di *slave* dan diubah ke xml lalu dikirimkan ke *recovery manager master*. Jika pengiriman keseluruhan data *temporary query* di *slave* ke *recovery manager master* berhasil, maka *recovery manager master* akan menyimpan data tersebut ke *linked list*. Kemudian *middleware* mengirimkan perintah *execute* ke *recovery manager master* dimana perintahnya berisi mengeksekusi *query* di *linked list*. Jika pengiriman keseluruhan data *temporary query* di *slave* ke *recovery manager master* tidak berhasil, maka *middleware* tidak mengirimkan perintah *execute* ke *recovery manager master*. Setelah *recovery manager master* selesai mengeksekusi data di *linked list* maka *recovery manager master* akan mengirim perintah *commit* ke *middleware*, sehingga *middleware* akan melakukan pemindahan koneksi pada *middleware* dari *slave* kembali ke *master*. Koneksi *write* di *middleware* akan menulis secara normal kembali yaitu ke *master* dan ke *slave*. Sedangkan proses *recovery* pada basis data *slave* adalah *middleware* mengambil data dari tabel *temporary query* di *master* dan diubah ke xml lalu dikirimkan ke *recovery manager slave*. Jika pengiriman keseluruhan data *temporary query* di *master* ke *recovery manager slave* berhasil, maka *recovery manager slave* akan menyimpan data tersebut ke *linked list*. Kemudian *middleware* mengirimkan perintah *execute* ke *recovery manager slave* dimana perintahnya berisi mengeksekusi *query* di *linked list*. Jika pengiriman keseluruhan data *temporary query* di *slave* ke *recovery manager slave* tidak berhasil, maka *middleware* tidak mengirimkan perintah *execute* ke *recovery manager slave*. Setelah *recovery manager slave* selesai mengeksekusi data di *linked list* maka *recovery manager* akan mengirim perintah *commit* ke *middleware*, sehingga koneksi *write* di *middleware* akan menulis secara normal kembali yaitu ke *master* dan ke *slave*.

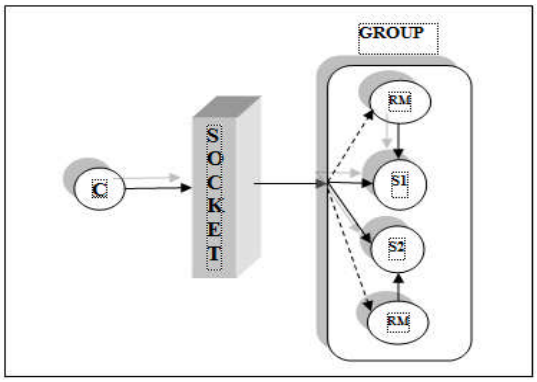
### *2.2.1 Perancangan model redundansi data*

Untuk menentukan model redundansi data, harus diketahui terlebih dahulu apa yang akan di replikasikan yakni fragmentasi data[6]. Penelitian ini menggunakan salah satu model fragmentasi pada basisdata yaitu *Non-Partitioned Database Replication* ditunjukkan pada Gambar 3. Pada model ini, fragmen pada setiap basisdata memiliki data yang sama. Data yang dimasukkan oleh *client* akan di replikasikan ke *master* dan *slave* secara serempak oleh *middleware* socket.




Gambar 3 Rancangan *non-partitioned database replication*

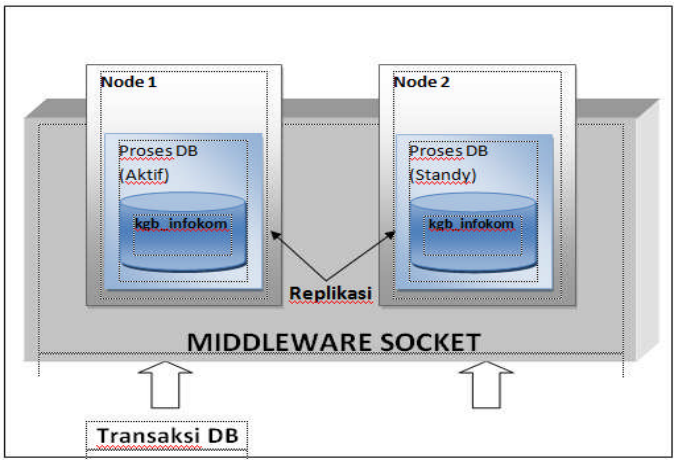
Penelitian ini menggunakan perancangan *file replication using a group.* Pada metode ini, semua proses pemanggilan sistem dalam bentuk WRITE ditransmisikan secara serempak ke kedua *server* yang ada, jadi salinan ditambahan dibuat pada waktu yang sama dengan yang aslinya[4]. Dengan menggunakan metode ini, konsistensi data pada kedua basisdata tetap terjaga. Gambar 4. menunjukkan rancangan replikasi yang dibuat menggunakan group. Saat *Client* (C) melakukan proses *write* maka data akan ditransmisikan secara serempak oleh *middleware* socket ke kedua *server* basisdata pusat yaitu basisdata *master*(S1) dan basisdata *slave*(S2). Garis putus-putus dari socket ke *recovery manager*(RM) dilakukan saat proses *recovery*.




Gambar 4 Perancangan replikasi file dengan group

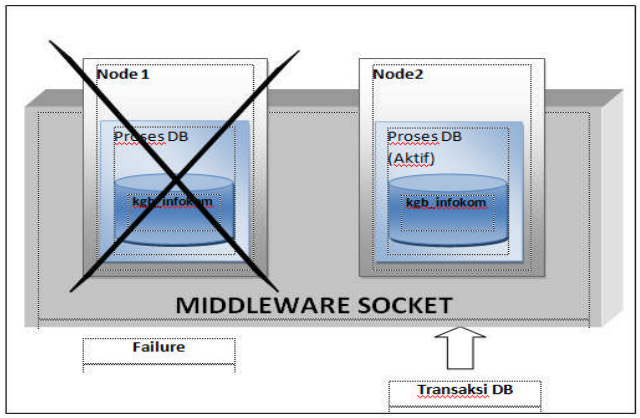
### *2.2.2 Perancangan model redundansi proses*

Model redundansi proses yang digunakan dalam penelitian ini adalah Aktif/Standby (2N). Setiap proses basisdata pada simpul aktif di *back up* pada simpul *standby*. Gambar 5 menunjukkan rancangan model redundansi proses dengan replikasi menggunakan *middleware* socket.




Gambar 5 Rancangan Active/Standby redundansi proses dengan replikasi

Ketika terjadi kegagalan pada proses aktif basisdata di *node* 2 yang berlokasi dibagian perencanaan, maka proses *standby* pada *node* 3 yang berlokasi di bagian keuangan akan mengambil alih dan menjadi proses aktif yang baru, dapat dilihat pada Gambar 6. Transaksi basisdata dipindahkan ke node 3 oleh *middleware* socket.

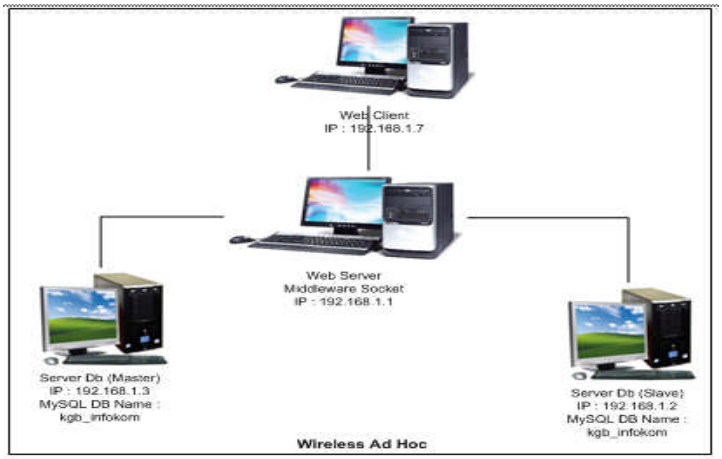



Gambar 6 Rancangan pergantian saat kegagalan primary aktif

Sementara proses transaksi dijalankan pada node 3, socket akan melakukan pengecekan terhadap basis data di node 2 secara terus-menerus. Apabila basisdata yang mengalami kegagalan pada *node* 2 sudah dapat bekerja kembali, socket akan merubah peran basisdata pada *node* 3 kembali menjadi proses *standby*, dan proses aktif kembali diperankan di node 2.

*2.3 Implementasi Sistem*

Sebelum bisa mengimplementasikan pembuatan aplikasi sistem KGB menggunakan *middleware* socket sebagai jembatan yang menghubungkan aplikasi *client* dengan kedua *database server*, harus terlebih dahulu membangun arsitektur jaringan yang akan menunjang pengimplementasian aplikasi tersebut. Pada Gambar 7 memperlihatkan arsitektur jaringan yang digunakan dalam membangun sistem.

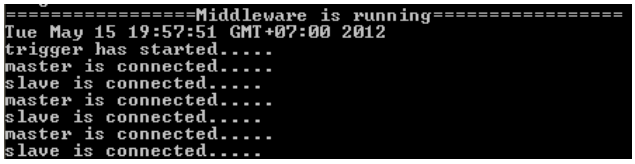



Gambar 7 Arsitektur jaringan sistem

## 3. HASIL DAN PEMBAHASAN

*3.1 Percobaaan dengan Master Slave Hidup*

Percobaan yang pertama dilakukan adalah dengan keadaan dimana kedua *server* basisdata pusat hidup atau terkoneksi dengan baik. Untuk operasi pembacaan, *coordinator* hanya akan melakukan pembacaan dari *server* basisdata pusat (*master*). Dan untuk operasi penulisan akan dilakukan pada kedua *server*. Gambar 8 menunjukan kondisi saat *middleware* dijalankan dan basisdata *master* dalam keadaan hidup. Status *middleware* socket dalam keadaan *server listening*.




Gambar 8 Tampilan *middleware* saat basisdata *master slave* hidup

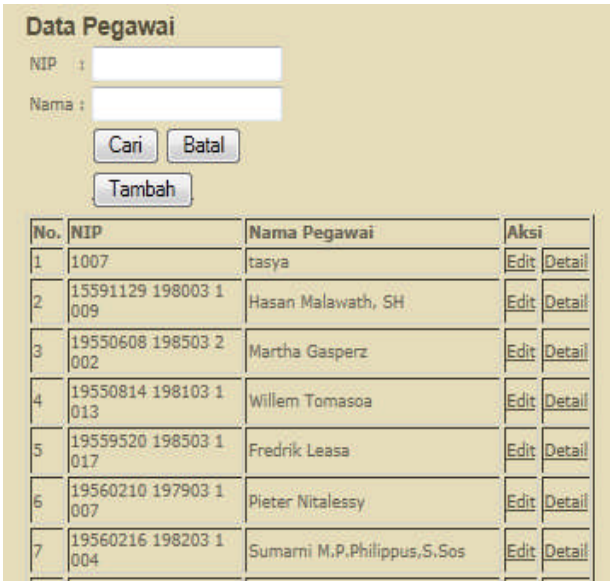
Dilakukan percobaan dengan melakukan *input* data pegawai. Kondisi awal tabel pegawai basisdata *server* kgb_infokom baik *master* maupun *slave* ditunjukkan pada Gambar 9.

| | nip | kode_instansi | nama_pegawai | tmt_cpns |
|---|---|---|---|---|
| | 15591129 198003 1 009 | I1 | Hasan Malawath, SH | 1980-03-01 |
| | 19550608 198503 2 002 | I1 | Martha Gasperz | 1985-03-01 |
| | 19550814 198103 1 013 | I1 | Willem Tomasoa | 1981-03-01 |
| | 19559520 198503 1 017 | I1 | Fredrik Leasa | 1985-03-01 |

Gambar 9 Kondisi awal tabel pegawai

Gambar 10 menunjukkan tampilan *form web client* saat akan melakukan *input* data pegawai. Data yang dimasukkan pada *form* ini akan diubah ke dalam format XML kemudian akan dikirim melalui *client s*ocket ke *middleware* socket. Gambar 11 menunjukkn tampilan untuk hasil pencarian data pegawai.


Gambar 10 Tampilan *form web client* untuk tambah data pegawai




Gambar 11 Tampilan *web client* untuk cari data pegawai

Gambar 12 menunjukkan tampilan *middlware* saat menerima kiriman data dari *web client.* Data berformat XML yang diterima *middleware* socket dari *web client* akan dimasukkan ke kedua basisdata *server*(*master* dan *slave*).

```
table name : pegawai
action : save
nip : 1007
kode instansi : I1
nama pegawai : tasya
master is connected.....
tmt cpns : 2012-04-18
Query : insert into pegawai (nip,kode_instansi,nama_pegawai,tmt_cpns) values ('1
007','I1','tasya','2012-4-18')
master is connected.....
master is connected.....
==================simpan data pegawai=====================
```

Gambar 12 Tampilan *middleware* saat proses *input*

Sehingga tabel pegawai pada basisdata *master* maupun *slave* akan menjadi seperti yang ditunjukkan pada Gambar 13.

| | nip | kode_instansi | nama_pegawai | tmt_cpns |
|---|---|---|---|---|
| | 1007 | I1 | tasya | 2012-04-18 |
| | 15591129 198003 1 009 | I1 | Hasan Malawath, SH | 1980-03-01 |
| | 19550608 198503 2 002 | I1 | Martha Gasperz | 1985-03-01 |
| | 19550814 198103 1 013 | I1 | Willem Tomasoa | 1981-03-01 |
| | 19559520 198503 1 017 | I1 | Fredrik Leasa | 1985-03-01 |

Gambar 13 Kondisi tabel pegawai setelah input data pegawai

*3.2 Percobaaan dengan Master Mati*

Saat basisdata *master* mengalami *error*, maka *middleware* socket akan memindahkan koneksi ke basisdata *slave* seperti yang ditunjukkan pada Gambar 14. Sementara trigger akan tetap melakukan pengecekan koneksi ke basisdata *master*.

```
master is connected.....
master is connected.....
change to slave ================
trigger detects master which is not connected....
trigger detects master which is not connected....
trigger detects master which is not connected....
trigger detects master which is not connected....
```

Gambar 14 Tampilan *middleware* saat pergantian koneksi ke *slave*

Saat proses input data pegawai pada Gambar 15 dilakukan sementara basisdata *master* dalam keadaan mati, maka data akan ditulis ke tabel pegawai pada *database slave* dan juga akan menulis *query sql statement* ke tabel *query* di basisdata *slave* untuk *recovery* nanti.


Gambar 15 Tampilan *input* data pegawai saat *master* mati

Gambar 16 menunjukkan tampilan *middleware* socket saat proses *insert* data ke tabel pegawai dan ke tabel *query* pada basisdata *slave*.


Gambar 16 Tampilan input data pada *middleware* saat *master* mati

Sehingga tabel pegawai pada basisdata *slave* akan berubah menjadi seperti pada Gambar 17. Sedangkan isi tabel pegawai pada basisdata *master* akan masih sama seperti pada Gambar 13. Sedangkan isi tabel *query* yang digunakan sebagai penyimpanan *temporary query* yang gagal dieksekusi pada basisdata *master* dapat dilihat pada Gambar 18.

| | nip | kode_instansi | nama_pegawai | tmt_cpns |
|---|---|---|---|---|
| | 1007 | I1 | tasya | 2012-04-18 |
| | 1008 | I1 | tacha | 2012-04-18 |
| | 15591129 198003 1 009 | I1 | Hasan Malawath, SH | 1980-03-01 |
| | 19550608 198503 2 002 | I1 | Martha Gasperz | 1985-03-01 |

Gambar 17 Kondisi tabel pegawai pada basisdata *slave*

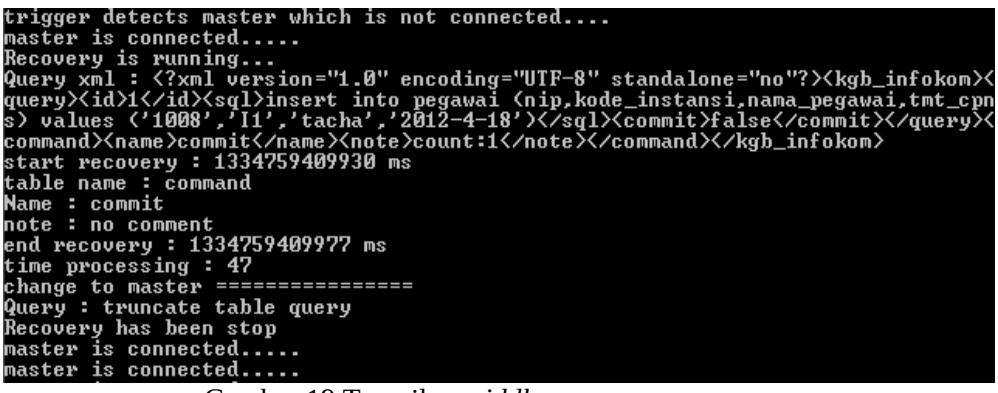
| | query_id | sql | commit |
|---|---|---|---|
| | 1 | insert into pegawai (nip,kode_instansi,nama_pegawa... | 0 |

Check All / Uncheck All With selected:

Gambar 18 Isi tabel *query*

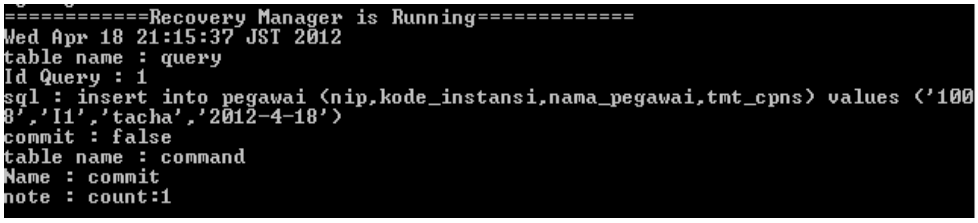
*3.3 Percobaaan saat master hidup kembali*

Saat *trigger* mendeteksi *database master* hidup kembali maka, proses *recovery* akan dijalankan. Setelah proses *recovery* selesai, maka *server* basisdata akan dikembalikan perannya ke basisdata *master*. Proses penulisan kembali ke keadaan awal, dimana penulisan dilakukan pada kedua *server* basisdata baik *master* maupun *slave* seperti yang ditunjukkan pada Gambar 19. Setelah itu isi tabel *query* pada basisdata *slave* dihapus.




Gambar 19 Tampilan *middleware* proses *recovery*

Gambar 20 menunjukkan tampilan pada *transaction manager* yang dibangun untuk menangani proses *recovery*. *Recovery manager* akan mengeksekusi data yang ada pada *Linked list* setelah mendapat perintah eksekusi dari *middleware* socket. Setelah proses eksekusi selesai dilakukan, maka *recovery manager* mengirim perintah *commit* ke *middleware* sehingga *middleware* akan memindahkan koneksi ke basisdata *master*. Kemudian, data pada *Linked list* dikosongkan.




Gamar 20 Tampilan *recovery manager*

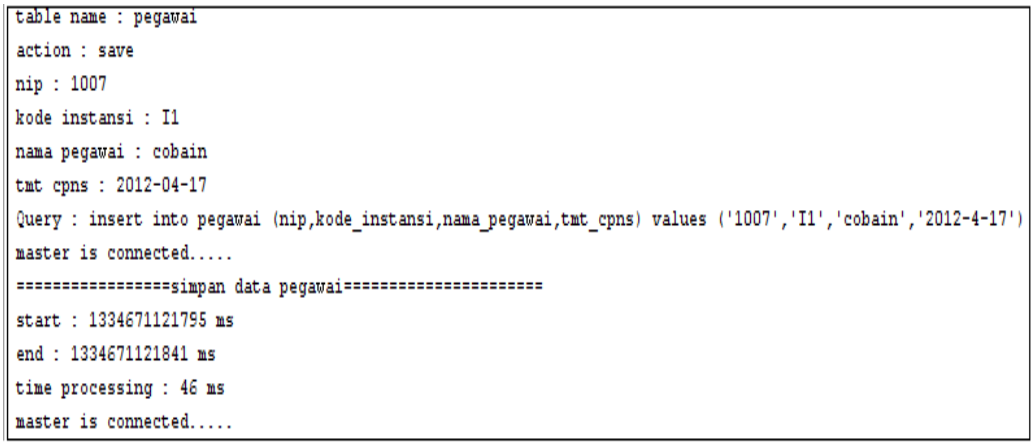
Pengujian dilakukan terhadap waktu proses *recovery* data pada tabel *historis_kgb*. Data KGB pegawai yang berada di tabel *historis_kgb* akan bertambah ketika pengguna melakukan *approve.* Tabel 1 memperlihatkan hasil pengujian untuk waktu proses *recovery* yang dihasilkan terhadap beberapa data yang berbeda.

Tabel 1 Hasil pengujian waktu proses *recovery*

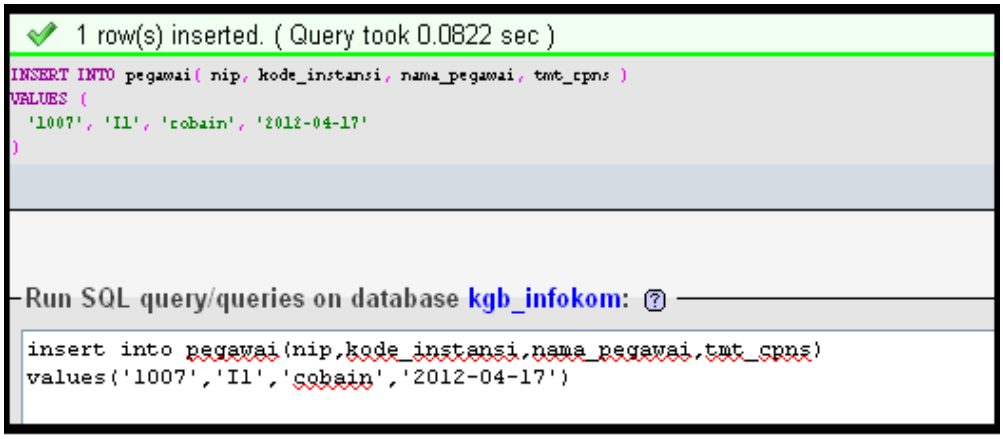
| Jumlah Data | Waktu Proses |
|---|---|
| 1 | 63 ms |
| 10 | 187 ms |
| 50 | 820 ms |
| 100 | 827 ms |

Dilakukan perbandingan dengan metode *master slave database replication*, dimana proses replikasi dilakukan langsung oleh basisdata *master* ke basisdata *slave*. Percobaan dilakukan dengan membandingkan waktu proses yang digunakan ketika melakukan proses *input* data pegawai. Data yang digunakan adalah data yang sama. Perbandingan dengan metode *master slave database replicaton*, membutuhkan konfigurasi pada dua buah komputer(komputer 1 dan komputer 2) yang menjadi basisdata *master* dan basisdata *slave*.

Gambar 21 menunjukkan hasil pengujian waktu proses untuk proses *input* data dengan *middleware* socket *replication.* Sedangkan Gambar 22 menunjukkan hasil pengujian waktu proses terhadap proses *input* data pada *master slave database replication*.




Gambar 21 Pengujian waktu proses untuk input data dengan *middleware* socket *replication*




Gambar 22 Pengujian waktu proses untuk *input* data
dengan *database master slave replication*

Perbandingan waktu proses antara metode replikasi dengan menggunakan Socket middleware yang dibangun dan metode replikasi basis data master slave dapat dilihat pada Tabel 2. Tabel 2 juga menunjukkan probabilitas yang diperoleh berdasarkan nilai rata-rata dari tiap proses yang diuji.

Tabel 2 Perbandingan metode replikasi dengan socket
dan *master slave database replication*

| **Perbandingan Waktu Proses (s)** | **Operasi Insert** | | **Operasi Update** | | **Operasi Delete** | |
|---|---|---|---|---|---|---|
| | **Socket** | ***Master Slave*** | **Socket** | ***Master Slave*** | **Socket** | ***Master Slave*** |
| rata-rata | 54.13 | 60.57 | 52.83 | 59.57 | 51.34 | 58.91 |
| Standar deviasi | 8.2 | 18.6 | 8.3 | 8.8 | 8.9 | 12.2 |
| **Perbandingan Probabilitas (%)** | | | | | | |
| Probabilitas > rata-rata | 40.517 | 45.62 | 40.517 | 40.905 | 41.294 | 43.644 |
| Probabilitas < rata-rata | 45.224 | 48.006 | 45.224 | 45.620 | 45.620 | 46.812 |
| Probabilitas antara rata-rata | 14.259 | 6.374 | 14.259 | 13.475 | 13.863 | 5.58 |

## 4. KESIMPULAN

Setelah melakukan implementasi dan melakukan pengujian terhadap penelitian ini maka terdapat beberapa hal yang dapat disimpulkan.
Kesimpulan dari penelitian ini adalah sebagai berikut:

1. *Middleware* socket yang dibangun dapat menangani masalah replikasi data pada basis data terdistribusi yang menggunakan model *Non-Partitioned Database Replication,* dengan metode perancangan proses replikasi yang digunakan adalah *file replication using a group*.
2. Aplikasi *middleware* socket yang dibangun mampu mengatasi permasalahan *fail-silent fault* pada sebuah *server* basis data. Sehingga sistem dapat tetap melanjutkan tugasnya dengan benar ketika terjadi kegagalan akses pada salah satu *server* basis data.
3. Dibawah kendali *middleware* socket, bila terdapat *query* penulisan yang gagal dieksekusi pada salah satu *server* basis data, maka *query* tersebut akan dicatat pada tabel *query* yang ada pada *server* basis data yang aktif sebagai *temporary query*. *Query* tersebut akan

digunakan untuk proses *recovery*. Proses *recovery* pada server basis data dilakukan dengan menggunakan *recovery manager*.

4. Berdasarkan hasil pengujian terhadap waktu proses replikasi dengan *middleware* socket, nilai rata-rata waktu proses operasi *insert*, *update* dan *delete* relatif stabil. Dengan probabilitas data yang memiliki waktu proses operasi *insert* diatas rata-rata sebesar 40.517%, dibawah rata-rata sebesar 45.224%, dan diantaranya sebesar 14.259. Probabilitas data yang memiliki waktu proses operasi *delete* diatas rata-rata sebesar 40.517%, dibawah rata-rata sebesar 45.224%, dan diantaranya sebesar 14.259. Sedangkan probabilitas data yang memiliki waktu proses operasi *insert* diatas rata-rata sebesar 41.294%, dibawah rata-rata sebesar 45.620%, dan diantaranya sebesar 13.863.
5. Telah diimplementasikan penerapan arsitektur basis data terdistribusi dengan socket pada aplikasi sistem kenaikan gaji berkala otomatis.

## 5. SARAN

Pada penelitian ini masih banyak kekurangan yang perlu diperbaiki. Maka diperlukan beberapa saran yang membangun agar penelitian ini memiliki nilai yang lebih bermanfaat di kemudian hari.

1. Penelitian ini menggunakan model *non-partitioned* yang *single fragment* dan *full* replikasi, namun diharapkan pada penelitian selanjutnya dapat dikembangkan menggunakan model *Partitioned Replicated Database* dan *Mixed Replicated Fragment*
2. Model redundansi proses yang digunakan dalam penelitian ini yaitu Aktif/Standby (2N), dimana basisdata *slave* harus selalu dalam keadaan *stand by* untuk menggantikan peran basisdata *master* saat basisdata *master* mengalami kegagalan, untuk itu dalam pnelitian selanjutnya diharapkan dapat dikembangkan menggunakan metode Aktif/S*Spare, N*Aktif, atau N*Aktif/S*Spare.
3. Pada penelitian ini tidak memperhatikan masalah keamanan data, untuk itu diharapkan pada penelitian selanjutnya dapat dikembangkan untuk masalah penanganan keamananan data.

## DAFTAR PUSTAKA

[1] Silberschatz, A., Korth, H.F. dan Sudarshan, S., 2002, *Database System Concept*, 4th ed, McGrow-Hill, Singapore.

[2] Weigle, M., 2004, Client/Server Computing & Socket Programming,. http://www.cs.clemson.edu/~mweigle/courses/cpsc826, diakses tanggal 27 Desember 2011.

[3] Özsu, T.M. dan Valduriez, P., 2011, *Principles of Distributed Database System*, 3th ed, Springer.

[4] Tanenbaum, A. S., dan Steen, M. V., 2002, *Distributed Systems Principles and Paradigms*, Prentice-Hall.

[5] Davenport, T, 1993, *Process Innovation: Reengineering work through information technology*. Harvard Business School Press, Boston.

[6] Drake, S., Hu, W,. McInnis, D. M., Skold, M., Srivastava, A., Thalmann, L., Tikkanen, M., Tobjornsen, O., Wolski, A., 2005, *Architecture of Highly Available Database*, Springer-Verlag.