Peningkatan Kinerja Siakad Menggunakan Metode *Load Balancing* dan *Fault Tolerance* Di Jaringan Kampus Universitas Halu Oleo

Alimuddin*[1], Ahmad Ashari[2]
[1]Universitas Haluoleo; Kampus Bumi Tridharma Anduonohu, Sultra
[2]Jurusan Ilmu Komputer dan Elektronika, FMIPA UGM, Yogyakarta
e-mail: ***[1]ali.pustikuh@gmail.com**, [2]ashari@ugm.ac.id

Abstrak

Penerapan sistem informasi akademik (siakad) berbasis web suatu perguruan tinggi sangatlah penting untuk meningkatkan pelayanan akademik. Penerapan siakad tersebut memiliki berbagai kendala terutama dalam menangani jumlah akses yang tinggi sehingga menyebabkan terjadinya overload. Selain itu bila terjadi kegagalan hardware atau software menyebabkan siakad tidak bisa diakses. Solusi dari permasalahan ini adalah penggunaan banyak server dimana beban yang ada didistribusi dimasing-masing server. Perlu metode dalam pendistribusian beban agar merata dimasing-masing server yaitu metode load balancing dengan algoritma round robin sehingga skalabilitas siakad tinggi. Sedangkan untuk menangani kegagalan server perlu fault tolerance agar ketersediaan siakad menjadi tinggi. Penelitian ini untuk membangun metode load balancing dan fault tolerance menggunakan perangkat lunak linux virtual server dan beberapa program tambahan seperti ipvsadm dan heartbeat yang memiliki kemampuan meningkatkan skalabilitas dan ketersediaan siakad. Hasil penelitian menunjukan bahwa dengan load balancing dapat meminimalkan response time sampai 5,7%, meningkatkan throughput sampai 37% atau 1,6 kali dan memaksimalkan pemanfaatan sumberdaya atau utilisasi sebesar 1,6 kali peningkatan, serta terhindar dari overload. Sedangkan ketersediaan yang tinggi diperoleh dari kemampuan server melakukan failover atau berpindah ke server yang lain bila terjadi kegagalan. Sehingga implementasi load balancing dan fault tolerance dapat meningkatkan layanan kinerja siakad dan terhindar dari kesalahan.

Kata kunci *—load balancing, fault tolerance, overload, sistem informasi akademik*

The application of academic information system (siakad) a web-based college is essential to improve the academic services. Siakad the application has many obstacles, especially in dealing with a high amount of access that caused the overload. Moreover in case of hardware or software failure caused siakad inaccessible. The solution of this problem is the use of many existing servers where the load is distributed in the respective server. Need a method of distributing the load evenly in the respective server load balancing is the method by round robin algorithm so high siakad scalability. As for dealing with the failure of a server need fault tolerance for the availability siakad be high. This research is to develop methods of load balancing and fault tolerance using software linux virtual server and some additional programs such as ipvsadm and heartbeat that has the ability to increase scalability and availability siakad. The results showed that with load balancing to minimize the response time to 5,7%, increase throughput by 37% or 1,6 times and maximize resource utilization or utilization of 1,6 times increased, and avoid overload. While high availability is obtained from the server's ability to perform failover or move another server in the event of failure. Thus implementing load balancing and fault tolerance can improve the service performance of siakad and avoid mistakes.

Keywords*—load balancing, fault tolerance, overload, academic information systems.*

Received May 21th,2015; Revised January 1st, 2016; Accepted January 20th, 2016

1. PENDAHULUAN

Universitas Halu Oleo (UHO) dalam meningkatkan pelayanan akademik telah menerapkan aplikasi sistem informasi akademik (siakad) *online* sejak tahun 2011. Layanan aplikasi siakad ini antara lain KRS *online*, pemberian nilai secara online, penjadwalan mata kuliah online dan pembayaran SPP mahasiswa dengan sistem *hots to host* dengan pihak bank. Selain itu universitas halu oleo bekerja sama dengan pihak telkom untuk akses jaringan internet dengan menyewa *bandwidth* sebesar 100Mbps. Jumlah transaksi data siakad meningkat dari tahun ketahun akibat dari peningkatan jumlah mahasiswa. Pada tahun 2014 jumlah mahasiswa UHO sebanyak 40.663 orang, data jumlah akses sistem ini rata-rata sekitar 240.399 perhari. Hal ini terjadi pada saat penawaran KRS mahasiswa, pengimputan nilai dan pembayaran SPP yang jadwalnya bersamaan. Dari data jumlah yang akses tersebut umumnya terjadi peningkatan pada saat minggu terakhir dari jadwal yang ditetapkan. Peningkatan jumlah akses (*request*) tersebut menyebabkan kinerja sistem menjadi sangat lambat karena terjadi kelebihan beban (*overload*) sehingga server tidak bisa diakses. Hal ini menimbulkan banyak keluhan baik dari mahasiswa, dosen dan seluruh sivitas akademika.

Permasalahan tersebut muncul diakibatkan karena siakad online ini dijalankan pada satu server atau server tunggal. Satu server mengalami peningkatan kinerja ketika adanya akses ratusan ribu dalam sehari. Masalah tersebut dapat diatasi dengan penggunaan banyak server dengan skalabilitas tinggi. Server-server tersebut berada dalam himpun server yang disebut dengan kluster server. Penerapan kluster server dapat menghilangkan hambatan dari kinerja CPU, meningkatkan keandalan dan ketersediaan aplikasi [1]. Sistem yang ada kemudian di distribusikan ke masing-masing server sehingga sistem menjadi terdistribusi.

Sistem terdistribusi menyediakan berbagi sumber daya sebagai salah satu keuntungan utama, yang menyediakan kinerja yang lebih baik dan keandalan daripada sistem tradisional lain dalam kondisi yang sama [2]. Penggunaan banyak server untuk sistem terdistribusi membutuhkan metode untuk mengatur pembagian beban dengan adil atau merata disetiap server. Berbagai penelitian telah dilakukan untuk mengatur pembagian beban kerja pada server yang klustering untuk meningkatkan kinerja sistem yaitu dengan menerapkan metode *load balancing*.

Penerapan *load balancing* dalam *web server* sangat penting dan dapat merupakan solusi yang tepat dan efektif untuk menangani beban server yang sibuk dan dapat meningkatkan skalabilitas pada sistem terdistribusi [3,4]. Server *load balancing* atau disebut juga dengan *director* mendistribusikan beban kerja ke *multiple* server dengan mempertimbangkan kapasitas dari setiap server untuk mengurangi terjadinya kegagalan server [5].

Permasalahan lain yang muncul ketika sistem telah didistribusikan di masing-masing server adalah bila adanya kesalahan atau kegagalan pada *hardware* atau *software* dari sistem tersebut. Sistem informasi akademik merupakan penunjang utama proses kegiatan akademik di perguruan tinggi, sehingga diharapkan sistem harus bebas dari kegagalan baik akibat dari *hardware* ataupun *software*. Beberapa penelitian telah dilakukan untuk menangani kegagalan tersebut yaitu dengan menerapkan *fault tolerance. Fault tolerance* dapat diterapkan untuk mendeteksi dan mentoleransi kerusakan secara *real-time* pada sistem terdistribusi [6]

2. METODE PENELITIAN

2.1. Metode dan Alat Ukur Load Balancing

Dalam penelitian ini, distribusi beban *load balancing* atau alokasi beban ke server siakad menggunakan algoritma *round robin*. Penggunaan metode ini didasarkan pada kemampuan server siakad yang akan digunakan merata. Keuntungan dari algoritma *round robin* adalah tidak memerlukan komunikasi antar-proses dan mempunyai kinerja terbaik di antara semua algoritma *load balancing* untuk aplikasi tertentu tujuan khusus[2]. Implementasi LVS-

NAT menggunakan algoritma *round robin* lebih handal dalam mengoptimalkan *throughput, utilization,* dan *response time* dari web server[2]. Level *load balancing* yang ditetapkan dalam penelitian ini adalah level IP dimana hal ini didasarkan pada fungsionalitas *load balancer* yang bertugas untuk meneruskan paket dari klien tanpa perlu mengetahui isi dari paket tersebut seperti pada level aplikasi, sehingga mengurangi *overhead.* Pendekatan level IP dapat digunakan sebagai solusi *load balancing* pada *mobile video*[7]. Alat pengujian *load balancing* dalam penelitian ini menggunakan program aplikasi *apachejmeter* untuk mendapatkan data *throughput, response time* dan utilisasi.

2.2. Metode dan Alat Ukur Fault Tolerance

Fault tolerance merupakan sistem dengan kemampuan untuk tetap beroperasi walaupun kondisi saat itu tidak mendukung (terjadi *fault* pada sistem) [8]. Saat ini *fault tolerance* menjadi bagian yang penting terus dikembangkan untuk meningkatkan reliabilitas sistem [9] dan menangani masalah, untuk menemukan teknologi yang paling kuat dan efisien [10]. Sistem *fault tolerance* yang diterapkan pada penelitian ini menggunakan pendekatan *hardware* dan *software failure,* untuk melihat kemampuan sistem siakad mentoleransi kesalahan.

2.3. Rencana Pengujian Load Balancing

Rencana pengujian setelah membangun sistem *load balancing* dan *fault tolerance* dilakukan untuk menguji kemampuan sistem yaitu dengan cara melakukan permintaan atau jumlah *request* yang berbeda yaitu 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000 pada waktu tertentu selama 10 hari untuk mendapatkan data *throughput, response time* dan utilisasi.

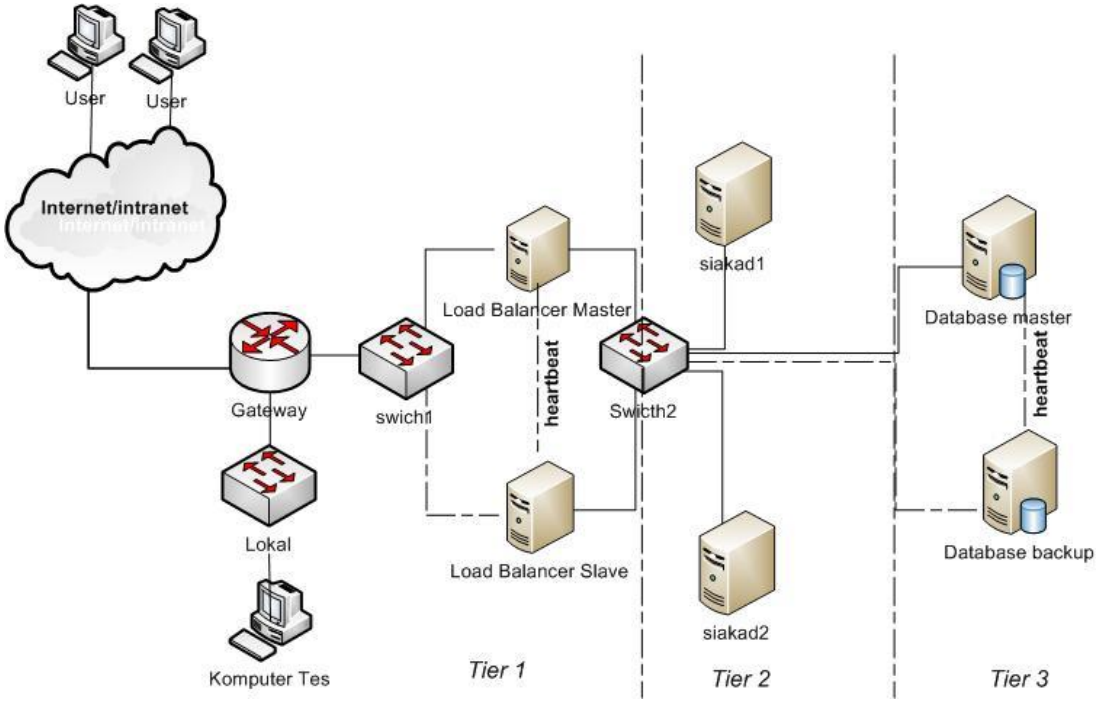
2.4. Rencana Pengujian Fault Tolerance

Rencana pengujian *fault tolerance* dilakukan untuk melihat ketersedian sistem terhadap adanya kegagalan atau kesalahan dari server baik secara *hardware* maupun *software*. Pengujian terhadap layanan sistem siakad agar memastikan sistem berjalan dengan baik. Adapun rencana pengujian *fault tolerance* sebagai berikut :

1. Rencana pengujian server *load balancing*
   a. Percobaan *load balancer master* dan *slave* hidup
   b. Percobaan *load balancer* master dimatikan
   c. Percobaan *load balancer* dihidupkan kembali
   d. Percobaan kedua *load balancer* dimatikan
2. Rencana pengujian *server siakad*
   a. Percobaan dengan *server siakad1* dan *server siakad2* hidup
   b. Percobaan *server siakad1* dimatikan
   c. Percobaan *server siakad1* hidup dan *server siakad2* dimatikan
   d. Percobaan *server siakad1* dan *server siakad2* diamatikan
3. Rencana pengujian server database
   a. Percobaan server *database master* dan *backup* hidup
   b. Percobaan server *database master* di matikan
   c. Percobaan server *database master* dihidupkan kembali
   d. Percobaan server *database master* dan *database backup* dimatikan
   e. Percobaan sinkronisasi data dari server *database master* ke *database backup*

2.5 Arsitektur Sistem

Arsitektur sistem *load balancing* dan *fault tolerance* yang diimplementasikan pada server aplikasi siakad di UHO memiliki struktur tiga lapis atau *three tier,* sebanyak 6 server seperti pada Gambar 1.



Gambar 1 Arsitektur *load balancing* dan *fault tolerance*

Tier pertama terdiri dari dua buah server load balancer yaitu sebagai master dan sebagai slave dikonfigurasi dengan IP 118.97.35.242 dan IP 192.168.0.161. Sedangkan load balancer slave dikonfigurasi dengan IP 118.97.35.241 dan IP 192.168.0.162. Kedua load balancer ini dihubungkan dengan 2 buah swich, swich1 dihubungkan dengan interfaces 1 dan swich2 dihubungkan dengan interfaces 2. Masing-masing load balancer di pasangkan heartbeat untuk membentuk IP virtual yaitu 222.124.222.61 sebagai IP dari http://siakad.uho.ac.id dan IP virtual 192.168.0.171 yang dipakai sebagai gateway pada server siakad di tier yang kedua.

Tier kedua adalah server siakad yang terdiri dari 2 server. Server siakad1 dikonfigurasi dengan IP 192.168.0.164 server siakad2 dikonfigurasi dengan IP 192.168.0.165. Kedua server siakad ini didalamnya menjalankan program aplikasi siakad.

Tier ketiga adalah storage atau penyimpanan database yang terdiri dari 2 server yaitu server database master dan server database backup. Server database master dikonfigurasi dengan IP 192.168.0.168 dan server database backup dengan IP 192.168.0.166. Kedua server database ini dipasangkan dengan heartbeat untuk membentuk IP 192.168.0.167 yang dipakai oleh aplikasi siakad di server siakad untuk mengakses database. Selain itu server database ini di lengkapi dengan program rsync untuk sinkronisasi database antara server database master dan server database backup dengan jadwal sinkronisasi yang ditentukan.

3. HASIL DAN PEMBAHASAN

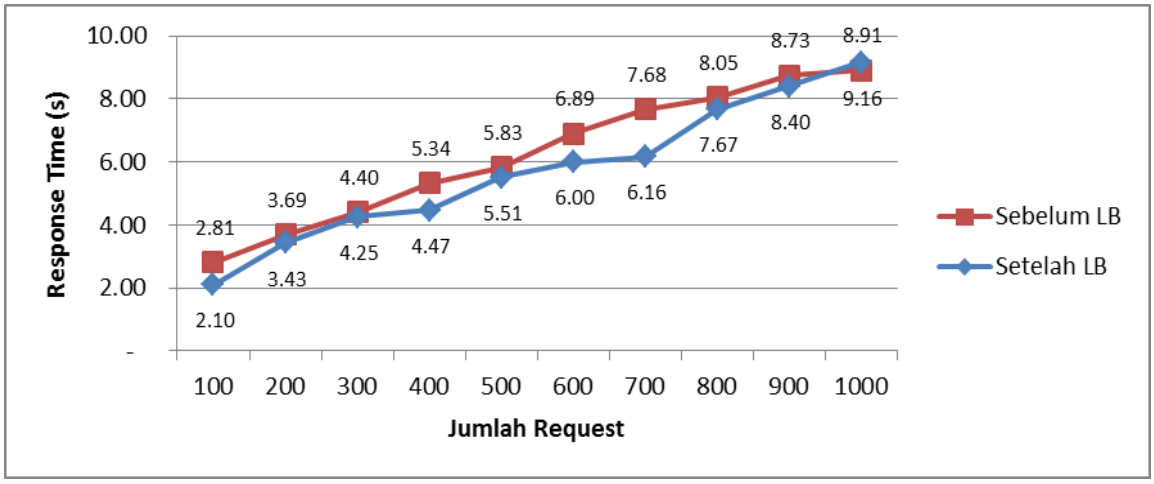
Hasil Pengambilan data trafik dilakukan dengan menggunakan bantuan perangkat lunak *ApacheJmeter* untuk mendapatkan nilai *response time*, *throughput* dan utilisasi. Data yang diperoleh akan dibandingkan antara sebelum dan setelah penggunaan *load balancing*. Sedangkan data *failover* diperoleh dari hasil uji *fault tolerance* yaitu dengan menguji server *load balancing*, server siakad dan server database.

3.1. Hasil Pengujian Load Balancing

Pengujian parameter *load balancing* dilakukan untuk melihat kemampuan server melayani sejumlah *request* dari pengguna dalam satuan waktu tertentu. Parameter yang diukur adalah *response time, throughput* dan utilisasi baik sebelum penerapan *load balancing* maupun setelah implmentasi *load balancing.*

3.1.1. Response Time

Response time atau waktu respon adalah total waktu yang dibutuhkan dari saat permintaan atau *request* akan menunggu untuk menerima jawaban. *Respon time* pada penelitian ini diukur untuk menentukan seberapa cepat aplikasi siakad merespon permintaan dari pengguna. Hasil rata-rata waktu respon sebelum *load balancing* dan setelah *load balancing* adalah berbeda. Nilai waktu respon sebelum *load balancing* rata-rata lebih tinggi dibanding setelah penerapan *load balancing* seperti terlihat pada Gambar 2.



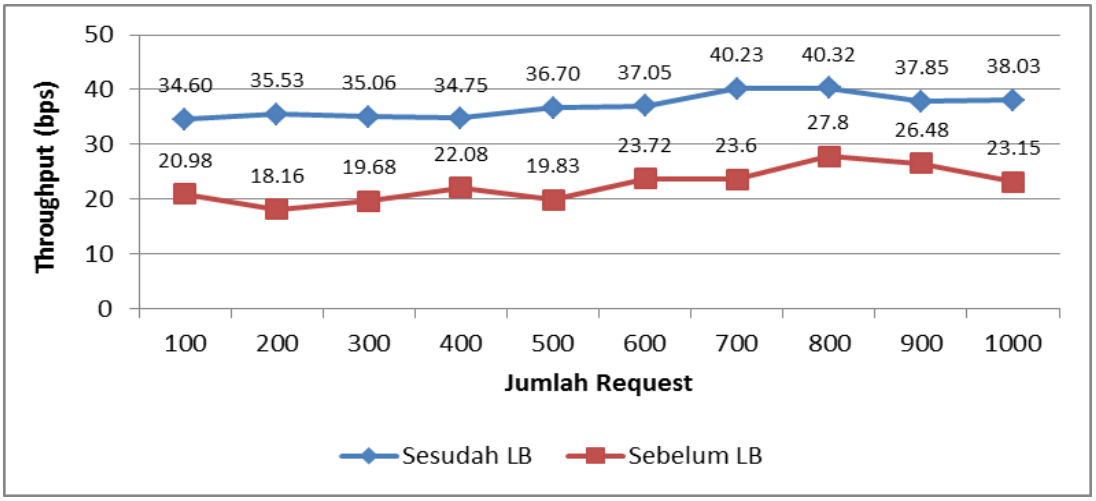
Gambar 2 Grafik *response time* sebelum dan setelah *load balancing*

Berdasarkan Gambar 2 terlihat bahwa setiap kenaikan jumlah *request* diikut juga dengan naiknya nilai waktu response ke siakad dimana waktu tertinggi sebelum *load balancing* adalah 8,91 dan terendah 2,81. Sedangkan setelah *load balancing* waktu tertinggi adalah 9,1 detik dan terendah adalah 2,10. Secara keseluruhan rata-rata menunjukan bahwa setelah *load balancing* waktu siakad merespon permintaan pengguna lebih cepat dibanding sebelum *load balancing.* Hal ini karena setiap *request* yang ada dibagi merata pada dua server siakad dibelakangnya, sehingga beban server menjadi berkurang dan kinerja siakad meningkat

.

3.1.2. Throughput

Throughput merupakan parameter kinerja yang memberikan gambaran mengenai besarnya data yang dapat dikirim dalam satu satuan waktu tertentu. Semakin banyak data yang dapat dikirim dalam satu satuan waktu tertentu maka semakin baik kinerja suatu sistem. *Throughput* pada penelitian ini digunakan untuk melihat *bandwidth* aktual atau jumlah *bit* yang ditransmisikan dalam satuan waktu *byte per second* (bps) baik sebelum *load balancing* dan maupun setelah penerapan *load balancing.* Hasil penelitian menunjukan bahwa nilai rata-rata *throughput* sebelum *load balancing* rata-rata lebih rendah dibanding setelah penerapan *load balancing* seperti terlihat pada Gambar 3.

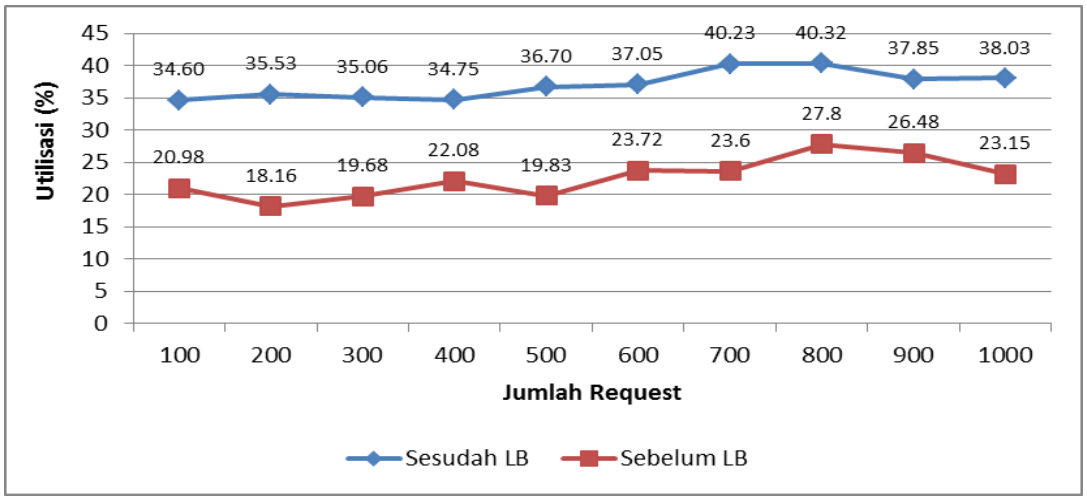


Gambar 3 Grafik *throughput* sebelum dan setelah *load balancing*

Dari total rata-rata seluruh *request* mulai 100 sampai 1000 menunjukan bahwa paket yang dapat di transmisikan lebih besar setelah *load balancing* yaitu 370,12 bps dibanding dengan sebelum *load balancing* yaitu sebesar 217,49 bps dengan selisih sebesar 144,67 bps. Selain itu dengan *load balancing* dapat meningkatkan nilai *throughput* sebesar 1,6 kali atau 37%. Besarnya paket data yang dapat ditransmisikan setelah *load balancing* disebabkan karena setiap paket yang masuk dapat didistribusikan secara merata pada kedua server siakad dibelakangnya dengan metode *round robin* sehingga dapat mengurangi *overload.* Nilai *throughput* yang besar ini menunjukan kinerja siakad meningkat.

3.1.3. Utilisasi

Utilisasi merupakan parameter yang menunjukkan seberapa besar persentase penggunaan *bandwidth* suatu *link*. Utilisasi pada penelitian ini diukur dengan menggunakan *tools stress ApacheJmeter* untuk menentukan seberapa besar persentase penggunaan bandwidth saat pengguna mengakses siakad. Hasil pengukuran utilisasi menunjukan bahwa nilai rata-rata nilai utilisasi sebelum *load balancing* lebih rendah dibanding setelah *load balancing* seperti terlihat pada Gambar 4.



Gambar 4 Grafik utilisasi sebelum dan setelah *load balancing*

Berdasarkan Gambar 4, rata-rata persentase penggunaan *bandwidth* menunjukan belum memberikan nilai maksimal, hal ini disebabkan tingginya beban atau *request* yang diujikan

dalam satuan waktu tertentu dimana hanya ditangani oleh dua server siakad sehingga bila ada *request* seperti itu maka server siakad perlu ditambah untuk memaksimalkan penggunaan bandwidth. Namun secara keseluruhan menunjukan bahwa rata-rata pemanfaatan bandwidth lebih tinggi setelah *load balancing* dibanding sebelum *load balancing*. Dengan *load balancing* persentase pemanfaatan *bandwidth* lebih baik yaitu sekitar 1,6 kali kenaikannya. Hal ini disebabkan karena beban yang ada dapat didistribusikan secara merata di dua server siakad dibelakangnya, sehingga beban menjadi berkurang dan kinerja siakad meningkat.

3.2. Hasil Pengujian Fault Tolerance

Fault tolerance atau toleransi kesalahan merupakan kemampuan sistem untuk untuk tetap beroperasi walaupun kondisi saat itu tidak mendukung atau terjadi *fault* pada sistem sedangkan *failover* merupakan sebuah mekanisme untuk mendeteksi server yang mengalami kegagalan *hardware* melalui sinyal *heartbeat* sehingga perannya berpindah ke server yang memiliki *heartbeat* aktif. Pada penelitian ini pengujian *fault tolerance* hanya dilakukan pada server setelah penerapan *load balancing*, sedangkan pada server tunggal atau sebelum penerapan *load balancing* tidak dilakukan karena jika dilakukan uji dengan mematikan server maka sudah tentu akan terjadi *down* atau server tidak akan dapat diakases. Ada tiga pengujian yang dilakukan yaitu : (1) menguji *load balancer master* dan *load balancer slave* (2) menguji server siakad dan server siakad 2 (3) menguji server database *master* dan *backup*.

3.2.1. Hasil Pengujian Failover Load Balancer

Laod balancer dalam penelitian ini adalah server yang berada di *tier* pertama yang bertugas menerima *request* dari pengguna, kemudian membagi *request* tersebut ke server siakad di *tier* yang kedua dengan metode *round robin*. Beberapa percobaan pengujian *load balancer* yaitu :

1. Percobaan *load balancer master* dan *slave* hidup

Pada pengujian ini semua server baik *load balancer*, server siakad dan server database dihidupkan. Kemudian sistem siakad diakses, disaat yang sama dilakukan monitoring *tabel virtual server* secara *real time* setiap 1 detik. Hasilnya adalah semua berjalan dengan normal *load balancer* dapat menjalankan tugasnya yaitu membagi beban yang masuk secara merata di kedua server siakad kemudian server siakad dapat menjalankan sistem siakad dengan mengambil data dari server database sehingga dapat dikatakan tingkat ketersediaan sistem siakad adalah 99,99% tersedia.

2. Percobaan *load balancer master* dimatikan

Pengambilan data dilakukan mulai dari mengakses sistem siakad dimana waktu dicatat pada pukul 09:27:09. Data waktu tersebut kemudian dihitung dengan data yang tercatat pada hasil *log* sistem. Berdasarkan data tersebut kemudian dihitung rata-rata sistem beroperasi sampai kegagalan terjadi selama 23 detik sedangkan waktu yang dibutuhkan sistem pulih atau beroperasi kembali selama 1 detik. Dari hasil tersebut kemudian dihitung persentase ketersediaan sistem siakad berdasarkan persamaan (1) :

$$\text{Availabilty} = \frac{\text{MTTF}}{\text{MTBF}} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \qquad (1)$$

$$= \frac{29}{29+2} = 0{,}9355 \text{ atau } 93{,}55\%$$

Tingkat ketersediaan yang diperoleh dari sistem setelah layanan *heartbeat* di *load balancer* master dimatikan adalah sebesar 93,55% tersedia yang artinya sistem sangat baik dalam mentoleransi kesalahan yang terjadi pada server *load balancer* sehingga sistem siakad masih tetap dapat beroperasi atau dapat diakses oleh pengguna.

3. Percobaan *load balancer master* dihidupkan kembali

Pengujian dengan layanan *heartbeat* pada *load balancer master* dihidupkan kembali kemudian dimonitoring untuk melihat perpindahan atau *failback request* yang ada pada *load balancer slave* ke *load balancer master*. Pengambilan data dilakukan mulai dari mengakses sistem siakad dimana waktu dicatat jam 10:29:18. Data waktu tersebut dikompilasi dengan data yang tercatat pada hasil *log load balancer master* dan *load balancer slave*. Hasil perhitungan rata-rata sistem beroperasi sampai kegagalan terjadi selama 28 detik sedangkan waktu yang dibutuhkan sistem pulih atau beroperasi kembali selama 1 detik. Dari hasil tersebut kemudian dihitung persentase ketersediaan sistem siakad berdasarkan persamaan (1):

Availabilty = $\frac{40}{40+2} = 0{,}9523$ atau 95,23%

Tingkat ketersediaan yang diperoleh dari sistem siakad setelah layanan *heartbeat* di *load balancer master* diaktifkan kembali adalah sebesar 95,23% tersedia yang artinya sistem lebih baik dalam mentoleransi kesalahan yang terjadi pada server *load balancer* sehingga sistem siakad masih tetap dapat beroperasi atau dapat diakses oleh pengguna.

4. Percobaan *load balancer master* dan *slave* dimatikan

Pengujian ini dilakukan dengan mematikan kedua *load balancer* baik master maupun slave. Hasil yang diperoleh menujukan bahwa sistem siakad sama sekali tidak bisa diakses atau tingkat ketersediaanya 0,00%.

3.2.2. Hasil Pengujian Failover Server Siakad

Server siakad pada penelitian ini adalah server yang berada di *tier* ke 2 yang akan menerima beban atau *load* dari server *load balancer* di *tier* pertama dan meminta database pada *tier* ketiga, yang jumlahnya ada dua yaitu server siakad1 dan server siakad2. Beberapa percobaan pengujian yaitu :

1. Percobaan server siakad1 dan server siakad2 hidup

Pengujian yang pertama dilakukan mulai dari mengakses sistem siakad yaitu pada waktu pukul 12:38:35 kemudian dalam saat yang sama memonitor koneksi di *load balancer master* secara *real time*. Hasilnya adalah *request* yang masuk di *load balancer master* di didistribusikan ke server siakad1 dan server siakad2. Hasilnya adalah pada waktu pukul 12:38:45 ada *request* yang diterima oleh *load balancing* IP 222.124.222.61 dengan *port* 80 *(http)* kemudian *request* tersebut didistribusikan ke server siakad1 dengan IP 192.168.0.164 dan server siakad2 dengan IP 192.168.0.165 secara merata dengan algoritma *round robin* (rr) yang masing-masing memiliki 6 koneksi yang aktif dan 12 koneksi tidak aktif pada server siakad1 8 koneksi tidak aktif untuk server siakad2, sehingga dapat dikatakan ketersediaan sistem siakad adalah 99,99% tersedia.

2. Percobaan server siakad1 dimatikan

Pengujian server siakad1 mati dan server siakad2 hidup dilakukan pertama memastikan kedua server siakad dapat menerima *load* atau beban dari *load balancer* melalui *monitoring secara real time* di *load balancer master*. Selanjutnya siakad diakses pada waktu jam 12:41:53. Pada waktu pukul 12:42:13 server siakad1 dimatikan dan waktunya tercatat di *file log* sistem operasi *linux*. Hasil monitoring *tabel virtual server* di *load balancer* terlihat bahwa pada waktu jam 12:42:15 koneksi aktif menjadi 0 dan koneksi yang tidak aktif menjad 3 sedangkan pada server siakad2 bertambah menjadi 12 koneksi aktif dan 1 detik berikutnya semua akses sudah terkoneksi dengan server siakad2 atau pada IP 192.158.0.165. Dari hasil tersebut kemudian dihitung persentase ketersediaan berdasarkan persamaan (1)

Availabilty = $\frac{20}{20+2} = 0{,}9091$ atau 90,91%

Tingkat ketersediaan yang diperoleh dari sistem siakad setelah server siakad1 dimatikan adalah sebesar 90,91% yang artinya sistem sangat baik dalam mentoleransi kesalahan yang terjadi pada server siakad1 sehingga sistem siakad masih tetap dapat beroperasi atau dapat diakses oleh pengguna.

3. Percobaan server siakad1 hidup dan server siakad2 dimatikan

Pengujian server siakad dengan menghidupkan server siakad1 dan mematikan server siakad2. Pertama-tama memastikan kedua server siakad hidup dan menerima *request* dari *load balancer master* Data diperoleh mulai dari mengakses siakad pada pukul 13:17:22 kemudian server siakad2 dimatikan pada waktu pukul 13:17:42 dan waktu yang tercatat di *file log* bahwa saat *reboot* pukul 17:42 sampai 17:44 atau selama 2 detik.

Dari hasil tersebut kemudian dihitung persentase ketersediaan berdasarkan persamaan(1)

Availabilty = $\frac{22}{22+2} = 0{,}9167$ atau 91,67%

Tingkat ketersediaan yang diperoleh dari sistem setelah server siakad2 dimatikan adalah sebesar 91,67% yang artinya sistem sangat baik dalam mentoleransi kesalahan yang terjadi pada server siakad2 sehingga sistem siakad masih tetap dapat beroperasi atau dapat diakses oleh pengguna.

4. Percobaan server siakad1 mati dan server siakad2 mati

Pengujian ini dilakukan dengan mematikan kedua server siakad baik server siakad1 maupun server siakad2. Hasil yang diperoleh menujukan bahwa sistem siakad sama sekali tidak bisa diakses atau tingkat ketersediaanya 0,00%.

3.2.3. Hasil Pengujian Failover Server Database

1. Percobaan server *database master* dan *backup* hidup

Pada pengujian ini kedua server database dalam keadaan aktif. Kemudian sistem siakad diakses pada waktu pukul 17:15:05 hasilnya adalah siakad dapat diakses data dapat ditampilkan artinya server database mampu melayani *server siakad* dalam menyediakan data dan *load balancer* dapat membagi beban yang masuk kepada kedua *server siakad* sehingga dapat dikatakan tingkat ketersediaanya adalah 99,99% tersedia.

2. Percobaan server *database master* dimatikan

Pengambilan data dilakukan mulai dari akses ke sistem siakad dari komputer test pada jam 07:34:53 dan setelah itu layanan *heartbeat* pada server database master di matikan kemudian membanding *file log* pada sistem database master dan *file log* pada sistem database backup. Dari data hasil *log* kemudian dihitung rata-rata sistem beroperasi sampai kegagalan terjadi selama 20 detik sedangkan waktu yang dibutuhkan sistem pulih atau beroperasi kembali selama 2 detik. Dari hasil tersebut kemudian dihitung persentase ketersediaan sistem siakad berdasarkan persamaan (1).

Availabilty = $\frac{20}{20+2} = 0{,}9091$ atau 90,91%

Tingkat ketersediaan yang diperoleh dari sistem setelah layanan *heartbeat* di server database master dimatikan adalah sebesar 90,91% yang artinya sistem sangat bagus mentoleransi kesalahan yang terjadi pada server database master sehingga siakad masih tetap dapat beroperasi atau dapat diakses oleh pengguna.

3. Percobaan server database master dihidupkan kembali

Pengujian dilakukan mulai dari saat mengakses sistem siakad dari komputer test pada pukul 10:28:01, selanjutnya layanan *heartbeat* pada server database master diaktifkan. Saat mengaktifkan layanan *heartbeat* tersebut tercatat dalam sistem file *log*. Dari data hasil *log*

kemudian dihitung rata-rata sistem beroperasi sampai kegagalan terjadi selama 37 detik sedangkan waktu yang dibutuhkan sistem pulih atau beroperasi kembali selama 2 detik. Dari hasil tersebut kemudian dihitung persentase ketersediaan sistem siakad berdasarkan persamaan (1):

Availabilty = $\frac{37}{37+2} = 0{,}9487 \text{ atau } 94{,}87\%$

Tingkat ketersediaan yang diperoleh dari sistem siakad setelah server database master dihidupkan adalah sebesar 94,87% yang artinya sistem mampu melakukan *failback* atau kembali mengambil perannya menggantikan server database *backup* sehingga sistem siakad masih tetap dapat beroperasi atau dapat diakses oleh pengguna.

4. Percobaan server *database master* dan *backup* dimatikan

Pengujian ini dilakukan mulai dari akses sistem siakad pada pukul 10:20:25 setelah itu service *heartbeat* dimatikan pada server database master dan *backup*. Hasil yang diperoleh menujukan bahwa sistem siakad sama sekali tidak bisa diakses atau tingkat ketersediaanya 0,00%.

5. Percobaan sinkronisasi data dari server database master ke database backup

Sinkronisasi database pada penelitian ini merupakan proses penyalinan data dari server *database master* ke server *database backup* yang telah ditetapkan jadwalnya pada tengah malam pukul 0.1.00. Pengamatan ini dilakukan berdasarkan nilai kapasitas (*size* atau ukuran) data dan waktu *update* terakhir atau *date modified* terhadap data. Hasil yang diperoleh adalah data pada *server database* master dan *database backup* adalah sama baik dari ukuran file dan waktu update, artinya proses sinkronisasi berjalan dengan baik.

4. KESIMPULAN

Dari penelitian peningkatan kinerja siakad dengan metode *load balancing* dan *fault tolerance* di jaringan kampus Universitas Halu Oleo didapatkan beberapa kesimpulan sebagai berikut:

1. Implementasi *load balancing* dapat memperkecil *response time* sampai 5,7%, meningkatkan nilai *throughput* sampai 37% atau 1,6 kali, mengoptimalkan pemanfaatan sumberdaya atau utilisasi sebasar 1,6 kali peningkatan serta terhindar dari kelebihan beban atau *overload*.
2. Pada saat *load balancer master* mengalami kegagalan perannya dapat digantikan oleh *load balancer slave* sehingga ketersediaan siakad tinggi yaitu 93,55% dan ketika *load balancer master* pulih dari kegagalan maka perannya dapat diambil kembali dari *load balancer slave* sehingga ketersediaan sebesar 95,23%. Hal yang sama juga ketika terjadi kegagalan pada server siakad1 maka *request* yang ada dapat dipindahkan ke server siakad2 dengan ketersediaan sebesar 90,91%, begitu pula sebaliknya ketika kegagalan terjadi pada server siakad2 *request* yang ada dapat dipindahkan ke server siakad1 dengan ketersediaan sebesar 91,67%. Demikian pula ketika kegagalan pada server *database master* perannya dapat berpindah ke server *database backup* dengan ketersediaan sebesar 90,91% dan setelah server *database master* pulih maka perannya diambil kembali dengan ketersediaan sebesar 94,87%.
3. *Load balancer* mampu membagi beban secara merata di kedua server siakad yaitu masing-masing 50 koneksi dengan pengujian 100 *request* dalam 1 detik. Sedangkan *fault tolerance* dapat meningkatkan ketersediaan siakad yang tinggi yang diperoleh dari kemampuan aplikasi *heartbeat* melakukan *failover* dan *failback* bila salah satu server di masing-masing *tier* mengalami kegagalan atau pulih kembali setelah kegagalan sehingga kinerja siakad meningkat.

4. Aplikasi *ApacheJmeter* memiliki kemampuan untuk membangun sejumlah *request* dalam satuan waktu tertentu sehingga sangat sesuai untuk menguji kemampuan siakad dalam melayani banyak pengguna.

5. SARAN

Dari hasil penelitian ini dapat dikemukakan beberapa saran yaitu :

1. Pada penelitian ini menggunakan *linux virtual server* dengan via NAT, perlu membangun *load balancing* dengan LVS via *Direct Routing* atau IP *Tunneling*.
2. Perlunya penelitian lebih lanjut tentang penerapan database terdistribusi pada kluster server.
3. Perlunya penelitian tentang penerapan *load balancing* yang diujicobakan pada jaringan yang berbeda ISP

DAFTAR PUSTAKA

[1] Kopper, K., 2005. *The Linux Enterprise Cluster-Build a Highly Available Cluster with Commodity Hardware and Free Software*. No Starch Press, Inc. San Francisco.

[2] Rabu, J.A., Purwadi, J., dan Raharjo, W.S., 2012. Implementasi Load Balancing Menggunakan Web Server Metode LVS-NAT. *Jurnal INFORMATIKA Vol. 8, No. 2.*

[3] Cardellini, V., Colajanni, M., dan Yu, P.S., 1999. Dynamic Load Balancing on Web-server Systems. *IEEE Internet Computing , vol. 3, no. 3, pp. 28-39.* USA.

[4] Lukitasari, D., dan Oklilas, A.F., 2010. Analisis Perbandingan Load Balancing Web Server Tunggal Dengan Web server Cluster Menggunakan Linux Virtual Server. *Jurnal Generic, Vol.5 No.2:2010., ISSN: 1907-4093. Fakultas Ilmu Komputer Universitas Sriwijaya.*

[5] Kopparapu, C., 2002. *Load Balancing Servers, Firewalls, and Caches*. Wiley Computer Publishing. John Wiley & Sons, Inc. New York Chichester Weinheim Brisbane Singapore Toronto.

[6] Zheng, Q. dan Shin, G.K., 1998. Fault-Tolerant Real-Time Communication in Distributed Computing Systems. *Journal. IEEE Transactions on Parallel and Distributed Systems, Vol. 9, No. 5.*

[7] Xingming, Z., dan Shaoxin, Z. 2012. One Load Balancing Solution for Mobile Video Surveillance System. *IEEE International Conference on Computer Science and Service System.*

[8] Jalote, P. 1994. *Fault Tolerance in Distributed Systems*. Prentice Hall, Englewood Cliffs, NJ.

[9] Ferdinando, 2004. Fault Tolerance in Real-time Distributed System Using the CT Library. Master's *Thesis*. Department of Electrical Engineering, Faculty EE-Math-CS. University of Twente. Belanda

[10] Haryono, Istiyanto, Harjoko, dan Putra., 2014. Five Modular Redundancy with Mitigation Technique to Recover the Error Module. *International Journal of advanced studies in Computer Science and Engineering IJASCSE,* Volume 3, Issue 2.