Kinerja *Metode Load Balancing dan Fault Tolerance* Pada *Server* Aplikasi *Chat*

***Sampurna Dadi Riskiono*[1], *Selo* Sulistyo[2], *Teguh Bharata Adji*[3]**

Departemen Teknik Elektro dan Teknologi Informasi
Universitas Gadjah Mada
Yogyakarta, Indonesia

sampurna.mti15@mail.ugm.ac.id[1]
selo@ugm.ac.id[2]
adji@ugm.ac.id[3]

Abstrak

Perkembangan aplikasi *social network* telah tumbuh begitu cepat dengan berbagai aplikasi yang didukung oleh piranti cerdas sebagai *platform* untuk menjalankannya. Salah satu aplikasi *social network* yang digunakan adalah aplikasi *chat*. Ketika jumlah pengguna yang mengakses layanan *chat* meningkat dan *server* tidak dapat mengatasinya tentu ini akan menjadi masalah yang mengakibatkan layanan *server* terhenti disebabkan adanya beban berlebih yang diterima oleh suatu *server* tunggal. Oleh karena itu diperlukan penelitian untuk merancang bagaimana membangun sistem *server* yang dapat menangani banyaknya permintaan layanan yang masuk agar beban dari *server chat* dapat diatasi. Hal ini bertujuan untuk meningkatkan pelayanan untuk setiap permintaan yang dikirim oleh pengguna. Salah satu solusi dari permasalahan tersebut adalah penggunaan banyak *server*. Perlu metode untuk mendistribusikan beban agar merata di masing-masing *server*, yaitu dengan menerapkan metode *load balancing* untuk mengatur pemerataan beban tersebut. Selanjutnya akan dilakukan evaluasi sebelum dan sesudah penerapan *load balancing*. Sedangkan untuk ketersediaan yang tinggi diperoleh ketika *server* memiliki kemampuan dalam melakukan *failover* atau berpindah ke *server* yang lain bila terjadi kegagalan. Sehingga penerapan *load balancing* dan *fault tolerance* dapat meningkatkan layanan kinerja aplikasi *chat* dan memperkecil kesalahan yang terjadi.

Kata Kunci: *load balancer, fault tolerance,* sistem *serverchat,* beban berlebih.

1. Pendahuluan

Membangun suatu *social network* dibutuhkan suatu sistem *server* yang dapat mengatasi sejumlah akses yang tinggi. Hal ini juga dikarenakan karakteritik user yang beragam dan komplek sehingga penerapan *server* tunggal tidak lagi dapat mengatasi hal tersebut. *Load balancing* pada *server* merupakan salah satu cara yang dapat digunakan untuk meningkatkan kinerja dan ketersediaan *server*, yaitu dengan membagi *permintaan layanan* yang datang ke beberapa *server* sekaligus, sehingga beban yang ditanggung oleh masing-masing *server* lebih ringan [1]. Dalam menanggulangi peningkatan kinerja ketika adanya akses yang begitu banyak, maka penggunaan banyak *server* dapat menjadi solusi untuk mengatasi hal tersebut. Himpunan dari banyak server disebut dengan kluster server. Sehingga dengan menerapkan kluster *server* maka dapat meningkatkan keandalan dan ketersediaan aplikasi [2][3]. *Server load balancing* atau disebut juga dengan nama *director* bertugas mendistribusikan beban kerja ke banyak *server* dengan mempertimbangkan kapasitas dari setiap *server* yang ada dan ini dapat mengurangi terjadinya kegagalan *server*[4]. Sistem yang ada kemudian didistribusikan ke masing-masing *server* sehingga sistem yang ada menjadi terdistribusi. Sistem terdistribusi menyediakan berbagai sumberdaya, sebagai salah satu keuntungan utamanya adalah dapat menyediakan kinerja yang lebih baik serta keandalan dari pada sistem tradisional yang lain dalam kondisi yang sama [5]. Penggunaan banyak *server* untuk sistem terdistribusi membutuhkan metode untuk mengatur pembagian beban secara adil atau merata pada masing-masing *server*. Banyak penelitian telah dilakukan terkait penerapan metode *load balancing* untuk mengatur pembagian beban kerja pada *server* dengan tujuan untuk meningkatkan kinerja sistem. Penerapan *load balancing* dalam web *server* sangat penting dan dapat merupakan sebuah solusi yang tepat dan efektif untuk menangani beban *server* yang sibuk serta diharapkan dapat meningkatkan skalabilitas pada sistem terdistribusi [6][3]. Permasalahan lain yang muncul ketika sistem telah didistribusikan di masing-masing *server* adalah bila adanya kesalahan atau kegagalan pada *hardwar e*atau *softwar*edari sistem tersebut. *Webserver*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 [7].

Dengan demikian, terdapat permasalahan yang dapat disarikan, yaitu :

1. Belum adanya pengukuran yang membandingkan baik pada faktor *throughput* atau *respon time* untuk *server* tunggal dan kluster *server* untuk aplikasi *chat*.
2. Belum ditentukannya sistem yang *fault tolerance* untuk *load balancing server chat.*

Dalam penelitian ini, metode *load balancer* dan *fault tolerance* akan digunakan untuk meningkatkan layanan kinerja dari *server chat* dengan membandingkan sebelum dan setelah penerapan *load balancing* serta ketersediaan sistem dengan adanya *fault tolerance*.

2. Metode

Dalam melakukan penelitian ini secara lebih detail metode yang akan dilakukan ditunjukkan seperti pada Gambar 1.

Gambar 1. Tahapan metode penelitian

Untuk mendukung kelancaran penelitian ini, hal pertama yang dilakukan adalah mencari dan mengumpulkan berbagai literatur yang mengarah kepada berbagai metode *load balancing, fault tolerance* dan aspek lain yang mendukung. Perancangan *load balancing* menggunakan berbagai metode yang selanjutnya akan dievaluasi. Pengujian *system load balancing* dan *fault tolerance* dilakukan dengan memberikan sejumlah permintaan dengan nilai yang sudah ditentukan. Pengolahan data dan analisis data dilakukan sesuai dengan skenario yang akan diberikan terhadap system *load balancing* dan *fault tolerance* dalam mengatasi kegagalan sistem.

2.1 *Social Network*

Social network atau jejaring sosial adalah layanan (*service*) yang memungkinkan individu untuk membangun *public* atau *semi-public profile* dan menentukan dengan siapa informasi dari individu tersebut akan dibagikan. Kata *network* atau *networking* dalam *social network* memiliki arti sebagai hubungan yang berarti bahwa *social network* memfasilitasi hubungan dan komunikasi antar individu melalui perangkat komputer [8].

2.2 Metode *Load Balancing*

Load balancing adalah sebuah metodologi jaringan komputer untuk mendistribusikan beban kerja di beberapa komputer atau kluster komputer untuk mencapai pemamfaatan optimal sumber daya, memaksimalkan *throughput*, meminimalkan waktu respon, dan menghindari kelebihan beban. Kluster *server* terdiri dari beberapa perangkat komputer yang saling terhubung dan bekerjasama sehingga mereka dapat dilihat sebagai satu sistem dalam banyak aspek, dan kluster komputer biasanya digunakan untuk meningkatkan kinerja dan ketersediaan dari beberapa komputer [6]. *Load balancers* dapat melakukan berbagai fungsi seperti *load balancing, traffic engineering,* dan *switching* cerdas trafik. *Load balancers* dapat melakukan pemeriksaan kesehatan pada *server*, aplikasi, dan konten untuk meningkatkan ketersediaan layanan dan pengelolaannya [9]. Dalam penerapan *load balancer* ada berbagai metode atau algoritma yang dapat digunakan untuk mendistribusikan beban kepada setiap *server* didalam sekumpulan *server*. Alat pengujian load balancing dalam penelitian ini menggunakan program aplikasi *httperf* untuk mendapatkan data *Throughput* dan *response time.*

2.3 Metode *Fault Tolerance*

Fault tolerance merupakan suatu sistem yang memiliki kemampuan untuk tetap beroperasi walaupun saat itu terjadi kondisi yang tidak mendukung (terjadi *fault* pada sistem) [10][6]. Saat ini *fault tolerance* menjadi bagian yang penting dan terus dikembangkan dalam meningkatkan reliabilitas system [11] dan menangani masalah, untuk menemukan teknologi yang paling kuat dan efisien [12]. Dalam penelitian ini system *fault tolerance* yang diterapkan menggunakan pendekatan hardware dan *software failure*, untuk melihat kemampuan sistem *chat* di dalam mentoleransi kesalahan.

2.4 Pengujian *Load Balancing*

Pengujian ini dilakukan setelah sistem *load balancing*dan sistem *fault tolerance* selesai

diterapkan. Hal tersebut dilakukan untuk menguji kemampuan sistem yaitu dengan cara melakukan permintaan atau koneksi jumlah permintaan yang berbeda yaitu dengan koneksi 1000, 2000, 3000, 4000 dan 5000 pada waktu tertentu sebanyak 10 kali dalam setiap rentang waktunya untuk mendapatkan nilai parameter dari *response time* dan *throughput*.

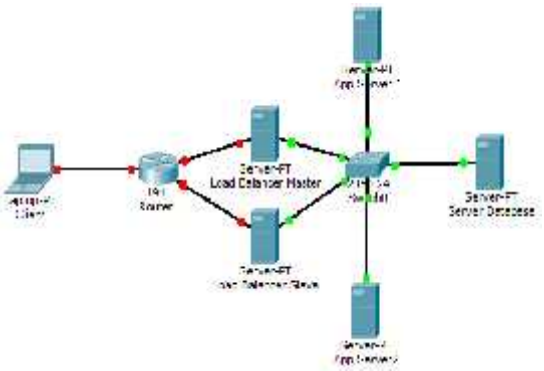
2.5 Pengujian *Fault Tolerance*

Pengujian *fault tolerance* dilakukan untuk melihat ketersedian sistem ketika terjadi kegagalan atau kesalahan dari *server* baik secara *hardware* maupun *software*. Pengujian terhadap layanan sistem *chat* agar memastikan sistem berjalan dengan baik. Adapun rencana pengujian *fault tolerance* sebagai berikut. Skenario pengujian server load balancing :

a. Kondisi *load balancer master* dan *slave* hidup
b. Kondisi *load balancer master* dimatikan
c. Kondisi *load balancer master* dihidupkan kembali
d. Kondisi kedua *load balancer* dimatikan

2.6 Arsitektur Sistem

Arsitektur dari sistem *load balancing* dan *fault tolerance* ini memiliki tiga bagian seperti diperlihatkan pada Gambar 2 yang mana terdiri dari *server load balancer master, server load balancer slave, server application 1, server application 2* serta *server database.*



Gambar 2 :Arsitektur *load balancer*dan *fault tolerance*

Pada bagian yang pertama terdiri dari dua buah server *load balancer* dimana ada yang berperan sebagai *master* dan sebagai *slave*. Kemudian untuk *load balancer* master dikonfigurasi dengan alamat IP 10.42.29.105. Sedangkan *load balancer slave* dikonfigurasi dengan IP 10.42.29.111. Kedua *load balancer* ini dihubungkan dengan sebuah *swich*. Masing-masing *load balancer* dipasangkan sebuah *keepalived* untuk membentuk IP *virtual* yaitu 10.42.29.109 sebagai alamat IP yang akan digunakan untuk mengakses aplikasi *chat*. Kemudian pada bagian kedua adalah *server* aplikasi *chat* yang terdiri dari 2 *server*. *Server* aplikasi *chat* 1 dikonfigurasi dengan alamat IP 10.42.29.114 sedangkan untuk *server* aplikasi *chat* 2 dikonfigurasi dengan alamat IP 10.42.29.112. Kedua *server chat* ini di dalamnya menjalankan program aplikasi *chatting*. Sedangkan untuk bagian yang ketiga adalah *storage* atau penyimpanan *database* yang terdiri dari sebuah *server database*. *Server database* ini dikonfigurasi dengan alamat IP 10.42.29.231.

3. Hasil dan Pembahasan

Dalam pengambilan data trafik dilakukan dengan menggunakan bantuan perangkat lunak httperf untuk mendapatkan nilai *response time* dan nilai *throughput*. Data yang diperoleh dari pengukuran selanjutnya akan dibandingkan antara sebelum penerapan *loadbalancing* dan setelah penerapan load balancing. Sedangkan untuk data *failover* diperoleh dari hasil uji *fault tolerance* yaitu dengan melakukan pengujian terhadap *server load balancing*.

3.1 Hasil Pengujian *Load Balancing*

Pengujian parameter *load balancing* ditujukan untuk melihat kemampuan server dalam melayani sejumlah permintaan yang datang dari pengguna dalam satuan waktu tertentu. Dalam pengujian tersebut parameter yang diukur adalah *response time* dan *throughput* hal ini dilakukan baik sebelum penerapan *load balancing* maupun setelah *load balancing* berhasil diimplementasikan. Pada rincian di atas, menunjukkan bahwa alamat IP 10.42.29.109 merupakan alamat yang dimiliki oleh *load balancer* yang berfungsi untuk meneruskan permintaan ke aplikasi *server*. Ketika pengguna meminta konten di *server* aplikasi, maka semua permintaan akan melewati *server load balancer*. Jadi pengguna tidak akan mengetahui *real server* mana yang melayani. Kemudian permintaan layanan tersebut akan dibagikan oleh *server load balancer* kepada *real server* yang ada di belakangnya dengan menyesuaikan pada algoritma penjadwalan yang telah disematkan. Selanjutnya setiap *real server* akan terhubung ke *server database* untuk menyesuaikan data yang akan digunakan, setelah itu permintaan akan di berikan kepada *client* melalui *server load balancer* kembali. Pengamatan dalam penelitian ini dilakukan dengan mengambil dua parameter yaitu *throughput* dan *response time*. Pengamatan ini bertujuan untuk membandingkan antara sebelum *load balancer* dan sesudah *load balancer* dibangun. Dalam hal ini, pada sisi klien akan dilakukan permintaan secara bertahap melalui *tool httperf* untuk melihat *throughput* dan *response time*. Hasil perbandingan *request* sebelum dan setelah penerapan *load balancing* dilakukan menggunakan *tool httperf* dalam

Tabel 1. Hasil perbandingan *request* sebelum dan setelah penerapan *load balancing*

No	Jumlah user	Tanpa Load Balancing		Dengan Load Balancing	
		Respon time (ms)	Throughput (KB/s)	Respon time (ms)	Throughput (KB/s)
1	1000-100/s	6,98	49,4	5,38	49,4
2	2000-200/s	5,24	98,8	5,5	98,8
3	3000-300/s	1145,4	121,4	4,99	148,2
4	4000-400/s	*Error*	*Error*	4,52	197,6
5	5000-500/s	*Error*	*Error*	23,55	246,76
Rata-rata Total		12,64	231,524	53,92	8,788

membuat sejumlah permintaan seperti ditunjukkan pada Tabel 1. Dimana permintaan layanan tersebut akan dilakukan secara bertahap. Dimulai dari jumlah koneksi dari koneksi 100/s untuk 1000 koneksi, 200/s untuk 2000 koneksi, 300/s untuk 3000 koneksi, 400/s untuk 4000 koneksi dan 500/s untuk 5000 koneksi yang dilakukan koneksi ke *server* secara bersamaan dalam satu waktu. Dari sini maka dapat dilakukan pengamatan serta membandikan kedua algoritma penjadwalan yang telah diimplementasikan, berdasarkan pada hasil yang telah diperoleh dari nilai *throughput* dan *response time*.

3.1.1 *Response Time*

Response time atau waktu respon adalah total waktu yang dibutuhkan dari saat permintaan atau permintaan akan menunggu untuk menerima jawaban [6]. Dalam penelitian ini pengukuran waktu respon dimaksudkan untuk menentukan seberapa cepat aplikasi *chat* merespon permintaan dari pengguna. Hasil rata-rata waktu respon sebelum *load balancing* dan setelah *load balancing* adalah berbeda.

Gambar 3 : Grafik espon time

Nilai waktu respon sebelum *load balancing* rata-rata lebih tinggi dibanding setelah penerapan *load balancing*. Berdasarkan Gambar 3 terlihat bahwa setiap kenaikan jumlah permintaan maka diikut juga dengan naiknya nilai waktu *response* dimana ketika jumlah koneksi diangka 3000 terlihat perbedaan yang sangat jauh. Sebelum *load balancing* mencapai 1145,4 ms sedangkan setelah *load balancing* waktu yang tercatat adalah 4,99 ms. Untuk koneksi 4000 dan 5000 terjadi *error* ketika dilakukan pengukuran pada pada sistem *server* tunggal sehingga nilai yang di hasilkan tidak *valid* namun dengan *load balncing server* masih dapat mengatasinya. *Error* tersebut dikarenakan kemampuan sistem *server* tunggal terbatas pada angka koneksi 3000. Secara keseluruhan rata-rata menunjukan bahwa setelah *load balancing* waktu *chat* merespon permintaan pengguna lebih cepat dibanding sebelum *load balancing*. Hal ini karena setiap permintaan yang ada akan dibagi merata pada ke dua *server chat* yang ada dibelakangnya, sehingga beban server menjadi berkurang dan kinerja *server chat* menjadi meningkat.

3.1.2 *Throughput*

Throughput merupakan parameter kinerja yang memberikan gambaran mengenai besarnya data yang dapat dikirim dalam satu satuan waktu tertentu [6]. Semakin banyak data yang dapat dikirim dalam satu satuan waktu tertentu maka semakin baik kinerja suatu sistem. *Throughput* pada penelitian ini digunakan untuk melihat *band width* actual atau jumlah bit yang ditransmisikan dalam satuan waktu *byte* per *second* (bps) baik sebelum *load balancing* maupun setelah load balancing [6]. Hasil pengukuran menunjukkan bahwa nilai rata-rata *throughput* sebelum *load balancing* lebih rendah dibanding setelah penerapan *load balancing* seperti terlihat pada Gambar 4. Berdasarkan Gambar 4 terlihat bahwa ketika jumlah

Gambar 4 :Grafik *Throughput* (Kb/Sec)

koneksi di angka 3000 terlihat perbedaan pada nilai *throughput*. Untuk koneksi 4000 dan 5000 terjadi *error* ketika dilakukan pengukuran pada sistem server tunggal sehingga nilai yang di hasilkan tidak *valid* namun dengan *load balncing server* masih dapat mengatasinya. *Error* tersebut dikarenakan kemampuan sistem *server* tunggal terbatas di angka koneksi 3000. Besarnya paket data yang dapat ditransmisikan setelah *load balancing* disebabkan karena setiap paket yang masuk dapat didistribusikan secara merata pada masing-masing *server chat* sehingga dapat mengurangi beban yang berlebih. Nilai *throughput* yang besar ini menunjukan kinerja *chat* menjadi meningkat.

3.2 Pengujian *Fault Tolerance*

Fault tolerance atau toleransi kesalahan merupakan kemampuan sistem untuk untuk tetap beroperasi walaupun terjadi kondisi yang tidak mendukung atau terjadi *fault* pada sistem. Sedangkan *failover* merupakan sebuah mekanisme untuk mendeteksi server yang mengalami kegagalan *hardware* melalui sinyal *keepalived* sehingga dapat melakukan perpindahan peran ke server yang memiliki *keepalived* aktif. Pada penelitian ini pengujian *fault tolerance* hanya dilakukan padaserver setelah penerapan *load balancing*, sedangkan pada *server* tunggal atau sebelum penerapan *load balancing* tidak dilakukan karena jika dilakukan uji dengan mematikan *server* maka dapat dipastikan server akan *down* atau *server* tidak dapat diakases.

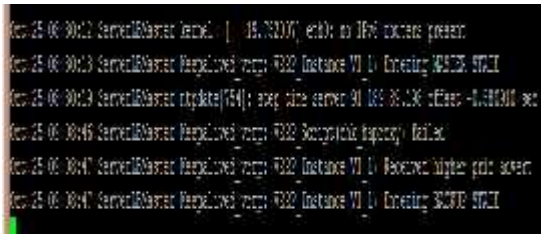
3.2.1. Pengujian *Failover Load Balancer*

Laod balancer dalam penelitian ini adalah server yang berada dibagian pertama yang bertugas menerima permintaan dari pengguna, kemudian membagi permintaan tersebut ke server *chat* di bagian yang kedua dengan metode *load balancing*. Beberapa percobaan pengujian *load balancer* yaitu :

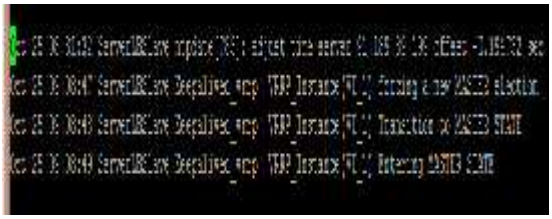
1. Percobaan load balancer master dan slave hidup.
   Pada pengujian ini semua server baik *load balancer*, server *chat* dan server *database* dihidupkan. Kemudian sistem *chat* diakses, di saat yang sama dilakukan monitoring *server* secara *real time*. Hasilnya adalah semua berjalan dengan normal, *load balancer* dapat menjalankan tugasnya yaitu membagi beban yang masuk secara merata di kedua *server chat* kemudian *server chat* dapat berjalan normal dalam melakukan komunikasi dengan mengambil data dari *server database* sehingga dapat dikatakan tingkat ketersediaan sistem *chat* adalah 99,99% tersedia.
2. Percobaan *load balancer master* dimatikan.
   Pengukuran dilakukan mulai dari mengakses sistem *chat* dimana waktu dimulai pada pukul 09:06:37. 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 129 detik sedangkan waktu yang dibutuhkan sistem pulih atau beroperasi kembali selama 2 detik seperti diperlihatkan pada Gambar 5, Gambar 6 dan Gambar 7. Dari hasil tersebut kemudian dihitung persentase ketersediaan sistem chat berdasarkan persamaan (1) :

MTBF = MTTR/ MTTR + MTTR

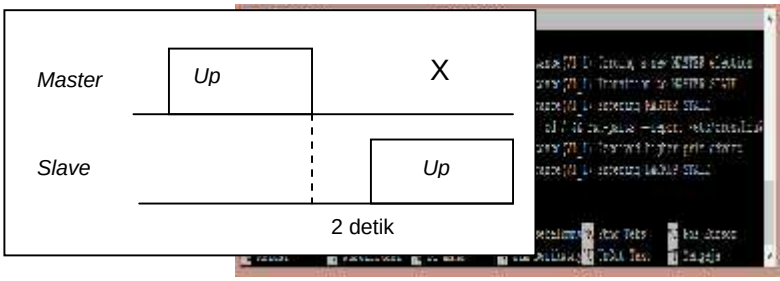
$$= \frac{129}{129 + 2} = 0{,}984 \text{ atau } 98{,}4\%$$



Gambar 5 : Monitoring keepalived pada load balancer master yang dimatikan



Gambar 6 : Monitoring keepalived pada load balancer slave



Gambar 7 : Monitoring *load balancer master* dimatikan

Tingkat ketersediaan yang diperoleh dari sistem setelah layanan *keepalived* di *load balancer* master dimatikan adalah sebesar 98,4% tersedia yang artinya sistem sangat baik dalam mentoleransi kesalahan yang terjadi pada server

load balancer sehingga sistem *chat* masih tetap dapat beroperasi atau dapat diakses oleh pengguna.

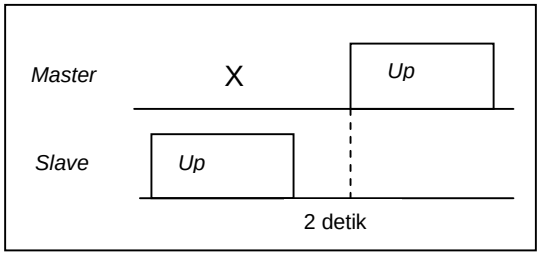
3. Percobaan *load balancer* master dihidupkan kembali.
   Pengujian dengan layanan *keepalived* pada *load balancer master* dihidupkan kembali kemudian dimonitoring untuk melihat perpindahan atau *failback* permintaan yang ada pada *load balancer slave* ke *load balancer master*. Pengambilan data dilakukan mulai dari mengakses sistem *chat* dimana waktu dicatat pukul 09:16:25. 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 65 detik sedangkan waktu yang dibutuhkan sistem pulih atau beroperasi kembali selama 2 detik seperti di perlihatkan pada Gambar 9, Gambar 10 dan Gambar 11. Dari hasil tersebut kemudian dihitung persentase ketersediaan sistem *chat* berdasarkan persamaan (1):

   MTBF = MTTR/ MTTR + MTTR

$$= \frac{65}{65 + 2} = 0{,}97 \text{ atau } 97\%$$

Gambar 8 :*Monitoringkeepalived* pada *load balancer master* yang dihidupkan kembali

Gambar 9 : Monitoring *keepalived* pada *load balancer slave*



Gambar 10 : Monitoring *load balancer master* aktif kembali

Tingkat ketersediaan yang diperoleh dari sistem *chat* setelah layanan *keepalived* di *load balancer master* diaktifkan kembali adalah sebesar 97 % tersedia yang artinya sistem lebih baik dalam mentoleransi kesalahan yang terjadi pada server load balancer sehingga sistem *chat* masih tetap dapat beroperasi atau dapat diakses oleh pengguna.

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

4. Kesimpulan

Dari penelitian mengenai metode *load balancing* dan *fault tolerance* pada *server chat social network* didapatkan beberapa kesimpulan sebagai berikut:

1. Penerapan metode *load balancing* pada *server chat* dapat memperkecil nilai dari *response time* serta dapat meningkatkan nilai *throughput* dimana ketika permintaan di atas 3000 koneksi sistem masih dapat melayani permintaan dari pengguna dibandingkan tanpa *load balancing* dimana *server* hanya mampu melayani permintaan sampai dengan 3000 koneksi. Artinya sistem dapat terhindar dari kelebihan beban yang datang dari pengguna.

2. Sistem dengan *fault tolerance* telah berhasil di terapkan sehingga saat *load balancer master* mengalami kegagalan maka perannya dapat digantikan oleh *load balancer slave* dalam waktu 2 detik dan ketika *load balancer master* pulih kembali maka perannya dapat diambil kembali dari *load balancer slave* dalam waktu yang sama yaitu 2 detik sehingga ketersediaan sistem terus terjaga.

Ucapan Terima Kasih

Terima kasih kepada DSSDI UGM yang turut membantu dalam penelitian ini.

Daftar Pustaka

[1] Lukitasari, D., dan Oklilas, A.F., 2010. AnalisisPerbandingan 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.

[2] Kopper, K., 2005. The Linux Enterprise Cluster-Build a Highly Available Cluster

with Commodity Hardware and Free Software. No Starch Press, Inc. San Francisco.

[3] W. Tarreau, "Making Applications Scalable," no. September, pp. 1–18, 2006.

[4] Kopparapu, C., 2002. Load Balancing *Server*s, Firewalls, and Caches. Wiley Computer Publishing. John Wiley & Sons, Inc.New Y ork Chichester Weinheim Brisbane Singapore Toronto.

[5] 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.

[6] U. Haluoleo, K. Bumi, and T. Anduonohu, "Peningkatan Kinerja Siakad Menggunakan Metode Load Balancing dan Fault Tolerance Di Jaringan Kampus Universitas Halu Oleo," vol. 10, no. 1, pp. 11–22, 2016.

[7] Chang, H.S., Chang, Y.M., dan Hsiao, S.Y., 2014. Scalable Network File Systems with Load Balancing and Fault Tolerance for Web Services. The Journal of Systems and Software. 93.102–109. Elsevier.

[8] E. Mit, N. H. Borhan, and M. A. Khairuddin, "Need analysis of culture-based genealogy software for indigenous communities," *2012 IEEE Symp. E-Learning, E-Management E-Services, IS3e 2012*, pp. 61–65, 2012.

[9] S. Malik, "Dynamic Load Balancing in a Network of Workstation", 95.515 Research Report, 19 November, 2000.

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

[11] 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

[12] 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.