khazanah informatika

Jurnal Ilmu Komputer dan Informatika

PERANCANGAN DAN PENGUJIAN *LOAD BALANCING* DAN *FAILOVER* MENGGUNAKAN NGINX

Rahmad Dani[1], Fajar Suryawan[2]
[1]Program Studi Informatika
Universitas Muhammadiyah Surakarta
[2] Program Studi Teknik Elektro
Universitas Muhammadiyah Surakarta
*Fajar.Suryawan@ums.ac.id

Abstrak
Situs web dengan *traffic* yang tinggi dapat menyebabkan beban kerja yang berat di sisi server, yang pada gilirannya akan mengakibatkan turunnya kinerja server, bahkan kegagalan sistem secara keseluruhan. Salah satu solusi untuk mengatasi masalah tersebut adalah dengan menerapkan teknik *load balancing* dan *failover*. *Load balancing* merupakan teknologi untuk melakukan pembagian beban kepada beberapa server, memastikan tidak terjadi kelebihan beban pada salah satu server. Sementara itu, *failover* merupakan kemampuan suatu sistem untuk berpindah ke sistem cadangan jika sistem utama mengalami kegagalan. Dalam penelitian ini *load balancing* dengan teknik *failover* akan diimplementasikan pada sistem operasi Ubuntu. *Software* inti yang digunakan dalam penelitian ini adalah Nginx dan KeepAlived. Nginx akan berfungsi sebagai *load balancer*, sedangkan KeepAlived untuk mengimplementasikan teknik *failover*. Beberapa skenario telah disiapkan untuk menguji sistem *load balancing* yang telah dirancang. Pengujian dilakukan dengan menggunakan perangkat lunak JMeter. Berdasarkan pengujian yang telah dilakukan, sistem yang dirancang berhasil membagikan beban permintaan dan dapat terus bekerja walaupun terjadi kegagalan pada server *load balancer* ataupun kegagalan pada server *backend*. Selain itu, dalam beberapa pengujian, penggunaan *load balancing* terbukti mampu menurunkan waktu respons dan meningkatkan *thoughput* pada sistem sehingga mampu meningkatkan performa keseluruhan sistem. Mengacu pada hasil penelitian ini, sistem *load balancing* dan *failover* menggunakan Nginx dapat dijadikan salah satu solusi pada sistem web server dengan situs web yang memiliki *traffic* tinggi.

Kata Kunci: load balancing, jaringan, failover, nginx, keepalived.

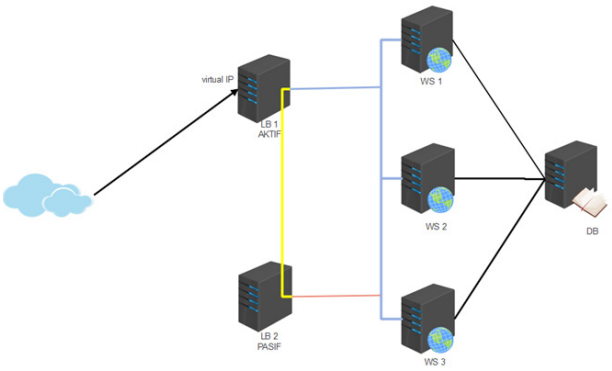
1. PENDAHULUAN

Seiring dengan perkembangan teknologi, jumlah penggunaan layanan web semakin meningkat. Hal ini menyebabkan situs-situs web popular memiliki jumlah *traffic* yang tinggi. Kegiatan atau acara tertentu juga dapat menyebabkan naiknya jumlah *traffic* web suatu organisasi. Contohnya, pada masa pendaftaran masuk kuliah atau sekolah *online*, pada masa KRS *online*, pada saat pemilu, atau pun *event-event* olahraga seperti piala dunia, olimpiade, dan *event-event* lain dapat menyebabkan peningkatan *traffic* web yang signifikan pada rentang waktu tersebut. Peningkatan jumlah *traffic* menyebabkan kerja server yang melayani permintaan menjadi semakin berat. Akibatnya performa server menurun dan sering terjadi gangguan pada layanan-layanan web tersebut. Jika tidak ditangani, hal ini dapat mengakibatkan sistem ataupun situs web tersebut mati/ *down*.

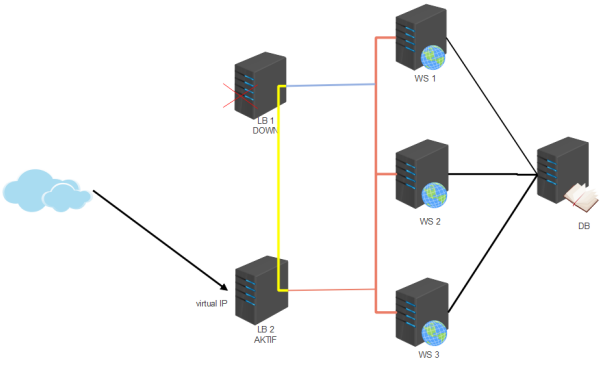
Salah satu solusi untuk menangani masalah tersebut adalah dengan menggunakan sebuah server yang andal dengan performa yang tinggi. Solusi lain yang dapat diterapkan adalah teknologi *load balancing*. Teknologi *load balancing* dapat membagi beban permintaan ke beberapa web server sehinga meringankan kinerja masing-masing server. Menurut Pandey, dkk (2015) *load balancing* merupakan aspek penting yang dapat mendistribusikan beban kerja ke banyak server secara optimal yang menghasilkan waktu tanggap yang baik dan meningkatkan tingkat kepuasaan pengguna, meningkatkan efisiensi penggunaan sumber daya, dan berdampak pada peningkatan performa secara keseluruhan.

Sedangkan menurut Friedrich, Gairing, dan Sauerwald (2012), *load balancing* diperlukan untuk efisiensi kegunaan sumber daya komputer dalam sistem paralel dan terdistribusi. Tujuannya untuk merelokasi beban sehingga nantinya tiap *node* menerima jumlah beban yang sama.

Beberapa penelitian yang telah dilakukan menunjukkan bahwa *load balancing* mampu meningkatkan performa server. Handoko Yoga Hartomo (2015) mengatakan bahwa teknik l*oad balance* bekerja dengan cara membagi beban yang diterima oleh server dan ketika salah satu mati, maka server lain akan melayani permintaan dari pengguna. Penelitiannya juga turut membuktikan hasil penggunaan dua *software load balancing* pound dan haproxy di mana keduanya sama-sama mampu meningkatkan kemampuan server namun terbukti haproxy sedikit lebih unggul.



Gambar 1. Sistem *load balancing* utama



Gambar 2. Sistem *load balancing* saat failover

Selain itu, Bayu Saputra (2012) juga menerapkan perancangan dan implementasi *load balancing* pada web server menggunakan haproxy, pengujian sistem dilakukan dengan membandingkan kinerja antara satu web server dengan sistem *load balancing* yang menggunakan dua web server. Hasilnya, beban kerja yang ditanggung pada sistem *load balancing* dibagi pada kedua server yang aktif sehingga kinerja server semakin cepat dan layanan tetap tersedia walaupun salah satu server *down*.

Dapat dikatakan bahwa penggunaan *load balancing* dapat mengatasi kegagalan yang terjadi pada sisi web server. Akan tetapi, jika *load balancing* mengalami kegagalan, seluruh layanan dapat terhenti. Oleh karena itu, diperlukan *failover* pada *load balancing* yang terpasang. Noviyanto, dkk (2015) menggunakan haproxy pada aplikasi web mendefinisikan bahwa *failover* merupakan kemampuan sebuah sistem untuk berpindah secara manual atau otomatis jika salah satu sistem mengalami kegagalan sehingga sistem lain menjadi *backup* bagi sistem yang mengalami kegagalan. Dengan cara ini sistem dapat terus berjalan walaupun sistem utama mengalami kegagalan.

Penelitian ini bertujuan untuk mengimplementasikan *load balancing* pada web server agar dapat meningkatkan kinerja sistem, serta mampu mengantisipasi kegagalan sistem melalui teknik *failover*. Manfaat dari penelitian ini adalah memahami konsep *load balancing* yang dipadu dengan teknik *failover* dengan menggunakan Nginx dan KeepAlived pada server berbasis Ubuntu.

2. Metode

Penelitian ini dilakukan secara eksperimental di Laboratorium Jaringan Komputer Fakultas Komunikasi dan Informatika Universitas Muhammadiyah Surakarta. Rancangan sistem *load balancing* dalam penelitian ini menggunakan 6 buah komputer yang berperan sebagai server. Dari 6 komputer tersebut, 2 di antaranya berperan sebagai *load balancer*, 3 berperan sebagai web server, dan 1 berperan sebagai server *database*. Gambaran umum dari rancangan sistem *load balancing* dapat dilihat pada Gambar 1.

Gambar 1 menunjukkan sistem *load balancing* yang menggunakan metode aktif-pasif. Dalam metode aktif-pasif ini, hanya satu *load balancer* aktif/master yang akan melayani permintaan yang masuk. *Load balancer* lain atau *load balancer* pasif/slave hanya ikut memonitor kondisi *frontend* dan *backend* server. Apabila terjadi kegagalan pada *load balancer* yang aktif, sistem akan melakukan *failover* sehingga *load balancer* pasif akan mengambil alih status aktif dari master sementara waktu hingga master berfungsi kembali. Gambar 2 menunjukkan apa yang terjadi setelah *failover*.

Apabila sistem telah diimplementasikan, maka sistem siap diuji dalam beberapa skenario yang telah dirancang.

2.1 Alat dan Bahan

Tahapan ini menentukan hal-hal yang dibutuhkan dalam proses penelitian seperti *hardware* dan *software*. Terdapat 7 komputer yang digunakan dalam penelitian ini, 6 sebagai server, dan 1 sebagai klien. Keenam komputer yang digunakan sebagai server tersebut memiliki spesifikasi *hardware* komputer sebagai berikut: Intel Core i3-2120 3.30GHz, 4GB DDR3, 500GB HDD dengan sistem operasi Ubuntu 12.10. Sedangkan komputer klien memiliki spesifikasi *hardware* sebagai berikut: Intel Core i3-2348 2.30GHz, 6GB DDR3, 500GB HDD dengan sistem operasi Windows 10. Untuk *software* yang digunakan, antara lain nginx, keepalived, jmeter, apache2, mysql, htop, saidar, nano, dan putty.

2.2 Implementasi Sistem

a. Pemberian IP Address
   Proses konfigurasi dimulai dengan pengaturan ip *address* tiap-tiap komputer. Pada *load balancer*, IP *address* yang digunakan adalah 192.168.52.9/10. Sedangkan pada web server akan diberikan IP 192.168.52.2/6/12. Untuk server *database*, akan diberikan IP 192.168.52.16. Karena terdapat 2 *load balancer*, diberikan 1 virtual IP yang dapat berpindah antar *load balancing*. Virtual IP ini berfungsi sebagai IP dari situs web/aplikasi web yang digunakan. IP *address* untuk virtual IP adalah 192.168.52.99. Pengaturan IP *address* pada Ubuntu dapat dilakukan dengan cara mengubah isi file $/etc/network/interface. Hasil perubahan dapat dilihat dengan menggunakan perintah $ifconfig.
b. Konfigurasi Load Balancing Server
   Pada kedua server *load balancing*, perlu dilakukan instalasi Nginx. Perintah instalasi Nginx melalui *repository* Ubuntu adalah $apt-get install nginx. Lalu dilakukan penambahkan file konfigurasi baru dengan perintah $sudo nano /etc/nginx/conf.d/lb.conf. Nginx lalu di-*restart* agar perubahan diterapkan. File konfigurasi tersebut memiliki isi seperti ditunjukkan pada Gambar 3.

```
upstream myapp1 {
        #least_conn;
        server 192.168.52.2:80 max_fails=3 fail_timeout=30s;
        server 192.168.52.6:80 max_fails=3 fail_timeout=30s;
        server 192.168.52.12:80 max_fails=3 fail_timeout=30s;
        }
    server {
        listen 192.168.52.99:80;
        access_log /var/log/nginx/lb.access.log;
        error_log  /var/log/nginx/lb-error.log error;

        location / {
            proxy_pass http://myapp1;

        }
}
```

Gambar 3. Konfigurasi lb.conf

```
#
# Log Martian Packets
#net.ipv4.conf.all.log_martians = 1
#
net.ipv4.ip_nonlocal_bind=1
```

Gambar 4. Penambahan slot ip.

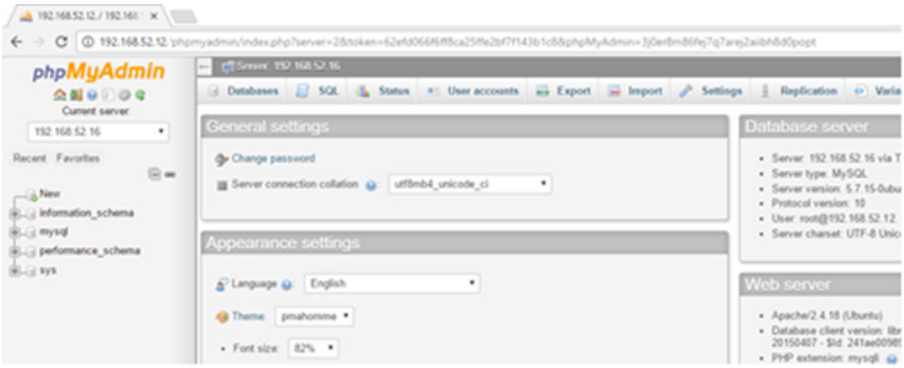
```
LogFormat "%v:%p %{X-Forwarded-For}i %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" vhost_combined
LogFormat "%{X-Forwarded-For}i %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%{X-Forwarded-For}i %l %u %t \"%r\" %>s %O" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

Gambar 5. Konfigurasi keepalived.conf

```
GNU nano 2.2.6        File: /etc/k
vrrp_script virtual_ip {
        script "pidof nginx"
        interval 2
}
vrrp_instance VI_1 {
        interface eth0
        state MASTER
        virtual_router_id 51
        priority 101
        virtual_ipaddress {
            192.168.52.99
        }
        track_script {
            virtual_ip
        }
}
```

```
GNU nano 2.2.6        File: /etc/keep
vrrp_script virtual_ip {
        script "pidof nginx"
        interval 2
}
vrrp_instance VI_1 {
        interface eth0
        state SLAVE
        virtual_router_id 51
        priority 100
        virtual_ipaddress {
            192.168.52.99
        }
        track_script {
            virtual_ip
        }
}
```

Gambar 6. Konfigurasi file apache2.conf

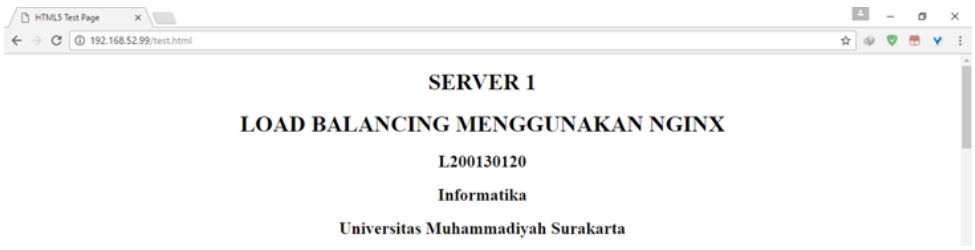


Gambar 7. Akses server *database* menggunakan phpmyadmin.

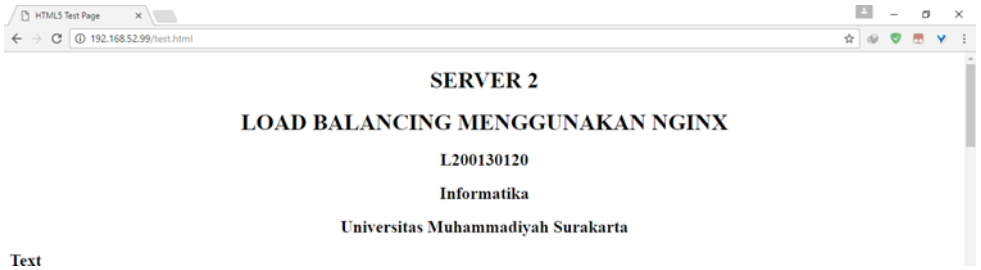
c. Konfigurasi Failover

Konfigurasi *failover* dimulai dengan menambahkan slot untuk ip virtual pada kedua server *load balancing*. Caranya dengan menambahkan baris net.ipv4.ip_nonlocal_bind=1 pada file $/etc/sysctl.conf. Agar perubahan diterapkan, masukkan perintah $sudo sysctl –p.

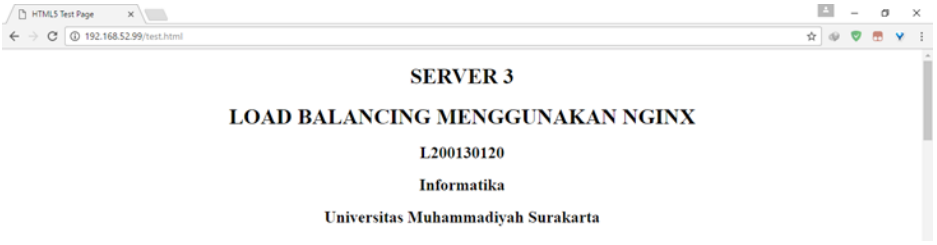
Selanjutnya, keepalived diinstal dengan perintah $sudo apt-get install keepalived. Digunakan perintah $sudo nano /etc/keepalived/keepalived.conf untuk membuat file konfigurasi baru. Konfigurasi tersebut disimpan, lalu keepalived di-*restart*. Terdapat perbedaan pada kedua file konfigurasi KeepAlived di kedua *load balancer*. Perbedaan tersebut adalah pada angka *priority*. *Priority* pada server *load balancing* master harus lebih tinggi dibandingkan pada server *load balancing slave*, tujuannya adalah agar kedua *load balancing* tidak memiliki ip virtual secara bersamaan.



Gambar 8. Server 1 melayani pengguna



Gambar 9. Server 2 melayani pengguna



Gambar 10. Server 3 melayani pengguna

Tabel 1. Pembagian beban

No	*Total Request*	Server 1	Server 2	Server 3
1	10	3	4	3
2	50	17	16	17

d. Konfigurasi Server Backend dan Server Database
Server *backend* dalam penelitian ini menggunakan apache2, php5, dan phpmyadmin. Agar *backend* dapat melihat ip klien, telah diedit kata %h menjadi %{X-Forwarded-For}i pada file konfigurasi $/etc/apache2/apache2.conf.

Pada server *database software* yang digunakan adalah mysql. Selain mudah, mysql dapat diakses secara remote melalui phpmyadmin.

2.3 Pengujian Sistem

Pada tahapan ini, sistem diuji menggunakan aplikasi JMeter untuk melakukan simulasi pemberian beban dari komputer yang bertindak sebagai klien. Saat melakukan pengujian, peneliti memonitoring keadaan server-server terutama server *backend* untuk mengetahui performa dari server *backend* dengan bantuan beberapa *software monitoring* seperti saidar, htop, top, dan juga beberapa perintah dasar pada linux seperti tail, awk, grep. Hasil-hasil dari uji coba tersebut akan dimuat dalam laporan beserta analisis dari peneliti.

Dalam pengujian, hasil pemasangan *load balancing system* ini dapat diakses melalui *browser* dengan alamat ip virtual yang terpasang yaitu http://192.168.52.99/test.html. Untuk melihat server mana yang sedang melayani *user*, maka pada sisi *backend* dapat dipasang konten berbeda.

3. Hasil Pengujian, Pembahasan, dan Diskusi

3.1 Pengujian Pembagian Beban

Pengujian ini dilakukan dengan cara mensimulasikan sejumlah klien yang mengakses sistem *load balancing* di mana setiap satu klien melakukan sekali *request*. Semua *request* tersebut, direkam oleh *log apache* pada *backend-backend* server. Data-data pada *log apache* tersebut dihitung dengan menggunakan perintah *grep*. Hasilnya dapat dilihat pada tabel berikut.

Sebelum setiap pengujian, isi *log apache* pada setiap *backend* server dihapus. Berdasarkan tabel 1, pada pengujian pertama menggunakan 10 *request*, urutan pelayanan *request* dimulai dari server 2, server 3, server 1 dan terus berulang sampai semua *request* terlayani. Hasilnya menunjukkan bahwa server 1 melayani 3 *request*, server 2 melayani 4 *request*, sedangkan server 3 melayani 3 *request*. Pada uji coba kedua dengan 50 *request*, urutan pelayanan *request* adalah server 3, server 1, server 2. Dimana server 1 melayani 17 *request*, server 2 melayani 16 *request* dan server 3 melayani 17 *request*. Dari hasil tersebut **terlihat bahwa *load balancer* membagikan beban secara merata secara berurutan sesuai dengan algoritma *round* robin**.

Tabel 5. Hasil uji dengan laman php

No	User	*Total request*	Sistem yang digunakan	Waktu respons rata-rata (ms)	*Throughput* (*request*/s)	*Error* (%)
1	400	10000	Tunggal	356	256,8	0
			LB	14	327,9	0
2	800	20000	Tunggal	1525	261,0	1,01
			LB	17	651,5	0
3	1200	30000	Tunggal	2840	218,1	4,80
			LB	349	760,0	0

Tabel 2. Ketersediaan saat terjadi kegagalan pada sisi *backend*

No Uji	*Total Request*	Server 1	Server 2	Server 3
1	10	5	5	down
2	10	5	down	5
3	10	Down	down	10

Tabel 3. Hasil pengujian *failover* pada *load balancer*

No	*Request*	Berhasil	*Error*	Keterangan
1	100	98	2	LB 1 ke LB 2
2	100	100	0	LB 2 ke LB 1

Tabel 4. Hasil uji dengan laman html

No	*User*	*Total Request*	Sistem yang Digunakan	Waktu Respons Rata-Rata (ms)	*Throughput* (*request*/s)	*Error*
1	400	10000	Tunggal	3	330,5	0
			LB	4	331,6	0
2	800	20000	Tunggal	3	657,1	0
			LB	7	658,3	0
3	1200	30000	Tunggal	93	907,0	0
			LB	239	805,7	0

3.2 Pengujian Ketersediaan Layanan

Pengujian selanjutnya yaitu dengan cara mematikan salah satu *backend* server yang ada. Pengujian ini dilakukan dengan memberikan 10 *request* kepada sistem yang sebelum pengujian dilakukan, satu atau dua *backend* server dimatikan. Tujuannya adalah untuk melihat apakah akan terjadi gangguan pada pelayanan oleh *backend* server.

Berdasarkan Tabel 2, pada pengujian pertama, server 3 dalam keadaan mati sehingga semua *request* dilayani oleh server 1 dan server 2. Pada pengujian kedua, server 2 dalam keadaan mati sehingga semua *request* dilayani oleh server 1 dan server 3. Pada pengujian ketiga, server 1 dan server 2 dalam keadaan mati sehingga semua *request* dilayani oleh server 3. Terbukti bahwa **apabila salah satu *backend* server mengalami kegagalan, hal ini tidak akan mengganggu sistem yang sedang berjalan sehingga pengguna tetap mendapatkan pelayanan dari sistem**.

Pengujian selanjutnya dilakukan untuk menguji *failover* pada *load balancing* sistem. Pengujian ini dilakukan dengan cara memberikan beban sebanyak 100 *request* dengan ketentuan 1 *request* perdetik selama 100 detik di mana pada pertengahan uji, server *load balancer* yang aktif akan sengaja dimatikan dan memaksa terjadinya *failover* ke server *load balancer* pasif. Berikut hasil percobaannya.

Uji coba pertama dari Tabel 3 dilakukan dengan cara mematikan server *load balancer* 1 saat sedang dilakukan pemberian *request*. Hasilnya menunjukkan bahwa ketika sedang terjadi perpindahan sistem dari server *load balancer* 1 ke server *load balancer* 2, *request* tidak dapat masuk karena sistem sedang *offline*. Pada percobaan pertama, sistem memerlukan waktu 5 detik untuk berpindah dari *load balancer* 1 ke *load balancer* 2 dan selama perpindahan tersebut terdapat 2 *request* oleh user yang tidak dilayani. Pada pengujian kedua, perpindahan sistem terjadi dari server *load balancer* 2 ke server *load balancer* 1. Pada pengujian ini, tidak terjadi *error* sama sekali yang menujukkan bahwa perpindahan sistem dari sistem cadangan kembali ke sistem utama tidak akan mempengaruhi jalannya sistem web. Pengujian ini membuktikan bahwa **sistem *load balancing* yang terpasang mampu melakukan *failover* untuk mengantisipasi kegagalan pada server *load balancer***.

3.3 Pengujian Kinerja Server

Pengujian Kinerja Server merupakan pengujian untuk mengamati performa *backend* server dalam keadaan diberi beban yang cukup banyak. Tujuan dari pengujian ini adalah untuk melihat perbandingan performa setelah menggunakan *load balancing* dengan sebelum menggunakan

Tabel 6. Hasil uji input data dengan php

No	User	*Total request*	Sistem yang digunakan	Waktu respon rata-rata (ms)	*Throughput* (*request*/s)	*Error* (%)
1	400	10000	Tunggal	185	288,6	0
			LB	172	285,6	0
2	800	20000	Tunggal	247	571,6	0
			LB	187	576,7	0
3	1200	30000	Tunggal	477	658,6	0.32
			LB	196	836,9	0

pada *load balancing*. Performa dihitung berdasarkan waktu respons yaitu waktu yang diperlukan sistem untuk melayani *request* dan juga *throughput* yaitu banyaknya *request* yang dapat dilayani sistem tiap detiknya. JMeter akan merekam kedua hasil tersebut.

3.4 Pengujian dengan Laman html Murni

Pada pengujian ini, dilakukan dengan cara memberikan sejumlah *request* pada sistem web yang didalamnya telah dipasang laman web html seberat 12 kB. Pengujian ini dilakukan dengan cara membandingkan hasil saat menggunakan *load balancing* dan hasil saat menggunakan server web tunggal. Berikut adalah hasil dari pengujian ini.

Berdasarkan Tabel 4, pengujian pertama menggunakan 400 *concurrent* user dengan total 10000 *request* yang diberikan dalam kurun waktu 30 detik. Terlihat bahwa rata-rata *response time* saat menggunakan server tunggal adalah 3 ms untuk melayani setiap *request* masuk. Hasil ini lebih baik jika dibanding dengan penggunaan *load balancing* yang mencapai 4 ms untuk tiap *request*-nya. Sedangkan *throughput* tidak banyak perbedaan di antara keduanya yaitu berkisar pada angka 331 *request* per detik. Pada pengujian kedua, dengan menggunakan 800 *concurrent* user dengan *Total request* sebanyak 20.000 yang diberikan dalam kurun waktu 30 detik, terlihat bahwa waktu rata-rata *request* dilayani oleh server tunggal adalah 3 ms sedangkan untuk *load balancing* membutuhkan waktu 7 ms. Untuk *throughput* keduanya memiliki jumlah yang relatif sama yaitu sekitar 657 *request* per detik. Pada pengujian ketiga dengan 1.200 *concurrent* user dan *total request* sebanyak 30.000 yang diberikan dalam kurun waktu 30 detik, menghasilkan waktu respons sebesar 93 ms pada server tunggal. Pada saat menggunakan *load balancing*, waktu respons yang didapatkan tidak lebih baik yaitu sebesar 239 ms. Untuk *throughput* pada server tunggal juga lebih baik yaitu sebesar 907 *request* per detik jika dibandingkan dengan *load balancing* yang sebesar 805,7 *request* per detik. Dalam pengujian ini, penggunaan *load balancing* pada sistem terbukti tidak memberikan dampak lebih baik.

3.5 Pengujian dengan Laman php

Pengujian ini dilakukan dengan memberikan sejumlah *request* kepada sistem web yang di dalamnya telah terpasang sebuah *file* php. File php tersebut akan melakukan perhitungan untuk mencari nilai pi (π) di sisi server dan hasilnya akan dikirim ke user sebagai respons dari permintaan yang dilakukan *user*. Pengujian ini membandingkan antara hasil dengan menggunakan sistem *load balancing* dan hasil menggunakan server web tunggal. Berikut ini merupakan hasil dari pengujian-pengujian tersebut.

Berdasarkan Tabel 5, pengujian pertama menggunakan 400 *concurrent* user dengan total 10000 *request* yang dikirim dalam kurun waktu 30 detik. Waktu respons rata-rata sistem untuk melayani *request* yang masuk saat menggunakan server tunggal adalah sebesar 356 ms. Sedangkan waktu respons rata-rata saat menggunakan *load balancing* menunjukkan hasil yang lebih baik yaitu sebesar 14 ms. *Throughput* saat menggunakan server tunggal hanya mencapai angka 256,8 *request* per detik jika dibanding dengan saat menggunakan *load balancing* yang sebesar 327,9 *request* per detik. Pada pengujian kedua dengan menggunakan 800 *concurrent* user yang melakukan *total request* sebanyak 20.000 yang diberikan pada kurun waktu 30 detik, hasilnya menunjukkan waktu respons rata-rata pada server tunggal sebesar 1.525 ms. Sedangkan saat menggunakan *load balancing*, waktu respons rata-rata sebesar 17 ms. *Throughput* pada server tunggal menunjukkan angka 261 *request* per detik sedangkan saat menggunakan *load balancing* hasilnya adalah 651,5 *request* per detik. Pada server tunggal 1,01% *request* dari *Total request* mengalami *error* sedangkan saat menggunakan *load balancing* tidak terdapat *request* yang mengalami *error*. Pengujian ketiga dengan menggunakan 1.200 *concurrent* user dan *Total request* sebanyak 30.000 yang diberikan dalam kurun waktu 30 detik menunjukkan bahwa waktu respons rata-rata saat menggunakan server tunggal adalah 2.840 ms. Sedangkan waktu respons rata-rata saat menggunakan *load balancing* adalah 349 ms. *Throughput* pada server tunggal menunjukkan bahwa sistem mampu melayani 218,1 *request* per detik, sedangkan saat menggunakan *load balancing*, sistem mampu melayani 760 *request* per detik. Pengujian ketiga pada server tunggal menunjukkan bahwa terdapat 4,80% *request* dari *total request* yang tidak terpenuhi sedangkan saat menggunakan *load balancing*, tidak ada *request* yang gagal terpenuhi. Hasil pengujian-pengujian dengan laman php menunjukkan bahwa **penggunaan sistem *load balancing* memiliki hasil lebih baik**.

3.6 Pengujian dengan laman php dan database

Pengujian ini dilakukan dengan cara memberikan sejumlah *request* pada sistem web untuk melakukan input data ke dalam *database* mysql melalui php. Dalam pengujian ini, data yang dimasukkan berupa 200 karakter *string*, *timestamp*, dan angka. Setiap *request* akan melakukan input data dua kali ke dalam *database*. Hasil dari pengujian ini adalah sebagai berikut.

Berdasarkan Tabel 6, hasil dari pengujian pertama dengan menggunakan 400 *concurrent* user dengan *total request* sebanyak 10.000 yang diberikan dalam waktu 30 detik menunjukkan bahwa waktu respons rata-rata menggunakan server tunggal adalah 185 ms sedangkan

Tabel 7. Hasil uji input gambar ke mysql

No	User	Total request	Waktu Respon (ms)	Throughput (request/s)	Error (%)
1	100	2000	12302	6,8	0
2	200	4000	22241	8	20,42
3	300	6000	3619	62,5	89,32

waktu respons rata-rata saat menggunakan *load balancing* adalah 172 ms. *Throughput* yang didapat ketika menggunakan server tunggal adalah 288,6 *request* per detik dan saat menggunakan *load balancing* memiliki hasil 285,6 *request* per detik. Pada pengujian kedua dengan menggunakan 800 *concurrent* user dan juga *total request* sebanyak 20.000 yang diberikan dalam kurun waktu 30 detik menunjukkan bahwa waktu respons rata-rata saat menggunakan server tunggal adalah sebesar 247 ms sedangkan saat menggunakan *load balancing* waktu respons rata-rata sebesar 187 ms. *Throughput* pada server tunggal mencapai angka 571,6 *request* per detik sedangkan *throughput* saat menggunakan *load balancing* hanya sebesar 576,7 *request* per detik. Pada pengujian ketiga, menggunakan 1200 *concurrent user* dengan total 30000 *request* yang diberikan dalam kurun waktu 30 detik menunjukkan bahwa waktu respon rata-rata saat menggunakan server tunggal adalah 477 ms sedangkan saat menggunakan *load balancing* adalah 196 ms. *Throughput* pada server tunggal menunjukkan angka sebesar 658,6 *request*s/detik sedangkan saat menggunakan *load balancing* hasil *throughput* menunjukkan angka 836,9 *request* per detik. Pada pengujian ketiga ini, saat menggunakan server tunggal terdapat 0.32% *request* dari *Total request* yang mengalami kegagalan, sedangkan saat menggunakan *load balancing*, tidak terdapat *request* yang mengalami kegagalan. Pada pengujian ini, **pemasangan *load balancing system* terbukti memberikan peningkatan kinerja**.

Pada pengujian selanjutnya, dilakukan dengan cara memasukkan gambar ke dalam *database* mysql. Gambar yang dimasukkan memiliki ekstensi jpg, tetapi pada mysql gambar akan memiliki format blob file. Ukuran gambar yang dimasukkan adalah sekitar 500 KB. Setiap *request* yang dilakukan oleh pengguna dalam pengujian kali ini akan memasukkan dua buah gambar. Perlu diketahui bahwa pengujian ini dilakukan setelah mysql dioptimasi dengan cara memperbesar jumlah koneksi yang mampu dilayani mysql tiap waktu dan juga memperbesar ukuran *log file* untuk innoDB. Jumlah koneksi tersebut diperbesar dari aturan *default* sebesar 150 menjadi 500, sedangkan ukuran *log file* diperbesar menjadi 128 MB. Hal ini dilakukan karena sebelum dioptimasi, pada sisi *backend* server sering mendapat peringatan *"to many connections"*, sedangkan pada sisi server *database* mendapat *error* dengan keterangan bahwa ukuran *log files* tidak cukup. Berikut merupakan hasil dari pengujian ini:

Berdasarkan hasil yang terlihat pada Tabel 7, pada pengujian pertama dengan 100 *concurrent user* dengan *total request* sebanyak 2000 *request* yang diberikan dalam waktu 30 detik, waktu respons rata-rata tiap *request* yang diberikan oleh *user* adalah sekitar 12.302 ms dengan *throughput* sebanyak 6,8 *request* per detiknya. Pada pengujian kedua, dengan 200 *concurrent* user yang memberikan *Total request* sebanyak 4000 dalam waktu 30 detik menghasilkan waktu respons sebesar 22.241 ms dengan *throughput* sebesar 8 *request* per detik. Dalam pengujian kedua, sekitar 20% *request* yang diberikan tidak/gagal dilayani oleh server. Pada pengujian ketiga dengan 300 *concurrent* user dengan *total request* sebanyak 6.000 dalam 30 detik memberikan hasil berupa waktu respons sebesar 3.619 ms dengan *throughput* sebesar 62,5 *request* per detik. *Request* yang gagal dilayani pada percobaan ketiga mencapai 89% dari *total request* yang diberikan. Pengujian input gambar ini membuktikan bahwa mysql tidak mampu melayani query berat dari 3 server *backend* di depannya walaupun telah dilakukan optimasi.

4. Penutup

Berdasarkan hasil pengujian dan pembahasan yang telah dilakukan dalam penelitian ini, dapat ditarik beberapa kesimpulan yaitu:

a. Berdasarkan pengujian, sistem yang dirancang dapat membagikan beban secara merata ke beberapa *backend* server baik dalam keadaan semua server normal atau pun saat terjadi kegagalan pada salah satu *backend* server.
b. Sistem yang dirancang dapat meningkatkan ketersediaan karena mampu melakukan *failover* saat terjadi kegagalan baik di sisi *load balancer* atau pun di sisi *backend* server.
c. Berdasarkan pengujian menggunakan jmeter yang telah dilakukan, nginx *load balancing* tidak meningkatkan waktu respons dan *throughput* pada *request* terhadap laman html. Akan tetapi saat melayani *request* laman php atau pun input data menggunakan php ke mysql, nginx *load balancing* mampu meningkatkan waktu respons dan *throughput* dengan baik.
d. Berdasarkan penelitian ini, penggunaan *load balancing* membuat sisi *database* menjadi lebih berat karena hanya menggunakan satu buah *database* sehingga saat menjalankan *request* yang memerlukan query yang berat, banyak *request* yang gagal.
e. Penggunaan *load balancing* dan *failover* nginx mampu menstabilkan dan menjaga performa sistem dapat dijadikan solusi untuk situs dengan *traffic* tinggi sehingga tidak terjadi kegagalan pada sistem dan aplikasi web yang terpasang pada sistem.

Saran bagi penelitian serupa yang akan datang adalah untuk dapat mengimplementasikan rancangan ke server dengan kondisi lingkungan yang nyata. Selain itu, penelitian selanjutnya disarankan untuk menggunakan lebih dari satu *database* untuk menopang sistem *load balancing* di depannya.

5. Daftar Pustaka

[1] Saputra, Ilham Bayu. (2012). Perancangan dan Implementasi *Load Balancing* Web Server Menggunakan *HAProxy*. Skripsi. Surakarta: Universitas Muhammadiyah Surakarta.
[2] Friedrich, T., Gairing, M., & Sauerwald, T.

(2012). *Quasirandom load balancing. SIAM Journal on Computing, 41*(4), 747-771. doi:http://dx.doi.org/10.1137/100799216

[3] Pandey, Shilipi. dkk (2015). *Load Balancing Techniques: A Comprehensive Study.* IJARCSMS: Volume 3 Issue 4.

[4] Noviyanto, Ari Budi. dkk. (2015). Perancangan dan Impementasi *Load Balancing Reverse Proxy* Menggunakan HAProxy pada Aplikasi Web. Jurnal JARKOM, vol 3 no 1.

[5] Hartomo, Handoko Yoga. (2015). Implementasi Web Server *Load balancing* pada Mesin Virtual. Skripsi. Surakarta: Universitas Muhammadiyah Surakarta.

[6] Nginx Documentation (2016). *Using Nginx as Load Balancing* http://nginx.org/en/docs/http/load_balancing.html (diakses pada 25 November 2016)

[7] Jmeter Documentation (2016). *Component Reference* http://jmeter.apache.org/usermanual/component_reference.html#Aggregate_Report (diakses pada 30 November 2016)