PENGUKURAN KINERJA ROUND-ROBIN SCHEDULER UNTUK LINUX VIRTUAL SERVER PADA KASUS WEB SERVER

Royyana Muslim Ijtihadie, Febriliyan Samopa
Jurusan Teknik Informatika, Jurusan Sistem Informasi
Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember
Kampus ITS, Jl. Raya ITS, Sukolilo – Surabaya 60111, Telp. + 62 31 5939214, Fax. + 62 31 5913804
Email : roy@its-sby.edu, iyan@its-sby.edu

ABSTRAK

Dengan meningkatnya perkembangan jumlah pengguna internet dan mulai diadopsinya penggunaan internet dalam kehidupan sehari-hari, maka lalulintas data di Internet telah meningkat secara signifikan. Sejalan dengan itu pula beban kerja server-server yang memberikan service di Internet juga mengalami kenaikan yang cukup signifikan. Hal tersebut dapat mengakibatkan suatu server mengalami kelebihan beban pada suatu saat. Untuk mengatasi hal tersebut maka diterapkan skema konfigurasi server cluster menggunakan konsep load balancing. Load balancing server menerapkan algoritma dalam melakukan pembagian tugas. Algoritma round robin telah digunakan pada Linux Virtual Server.

Penelitian ini melakukan pengukuran kinerja terhadap Linux Virtual Server yang menggunakan algoritma round robin untuk melakukan penjadwalan pembagian beban terhadap server. Penelitian ini mengukur performa dari sisi client yang mencoba mengakses web server.performa yang diukur adalah jumlah request yang bisa diselesaikan perdetik (request per second), waktu untuk menyelesaikan per satu request, dan throughput yang dihasilkan.

Dari hasil percobaan didapatkan bahwa penggunaan LVS bisa meningkatkan performa, yaitu menaikkan jumlah request per detik, menurunkan waktu menyelesaikan request, dan menaikkan throughput data yang ditransfer, sehingga data bisa diambil dalam waktu yang cepat..

***Kata kunci** : Linux Virtual Server, Load Balancing , Round-robin, IPv6, Kinerja*

1. PENDAHULUAN

Dengan meningkatnya perkembangan jumlah pengguna internet dan mulai diadopsinya penggunaan internet dalam kehidupan sehari-hari, maka lalu lintas data di Internet meningkat secara signifikan. Sejalan dengan itu pula beban kerja server-server service di Internet juga mengalami kenaikan yang cukup signifikan.hal tersebut dapat mengakibatkan suatu server mengalami kelebihan beban pada saat-saat tertentu, misalnya pada web server yang banyak dikunjungi oleh user di Internet.

Untuk mengatasi masalah beban berlebih pada server, terdapat dua solusi yang bisa dilakukan, yang pertama adalah menggunakan single server yang kemampuannya di-*upgrade* sehingga bisa menangani problem tersebut, yang kemudian pasti akan juga mengalami masalah kelebihan beban. Hal tersebut akan mengharuskan server tersebut kembali di-*upgrade*.

Solusi kedua adalah menggunakan multi server.multi server menggunakan *cluster* server-server yang bekerja saling mendukung. Ketika beban suatu server meningkat, maka untuk mengatasi kelebihan beban, bisa ditambahkan suatu server baru yang kerjanya sama untuk menangani *request* tambahan tersebut. Akan tetapi perlu diperhatikan bahwa dengan bertambahnya jumlah server yang masuk dalam *cluster* tersebut seharusnya tidak dirasakan oleh pengguna (*seamless*).

Virtual server merupakan suatu server dengan skalabilitas dan availabilitas tinggi yang dibangun dari sekumpulan server (*server cluster*). Arsitektur dari *cluster* server adalah transparan terhadap *end-user* sehingga *end-user* hanya akan melihat adanya mesin tunggal saja.

Virtual server melakukan distribusi beban dengan menggunakan algoritma *scheduling*. Algoritma *scheduling* yang umum digunakan adalah algoritma *Round-robin*, yang akan membagi rata beban kepada semua server yang menjadi anggota *cluster*. Linux Virtual Server telah banyak digunakan dalam banyak sistem seperti pada Janet[XXX].

Dalam suatu sistem yang menjadi infrastruktur sebuah service, maka diperlukan kinerja yang baik dalam aspek throughput (jumlah data yang bisa ditransfer dari service tersebut, per detik). Dengan throughput yang lebih besar, maka penggunaan LVS terbukti akan meningkatkan performa dari sistem secara keseluruhan. sehingga biaya yang dikeluarkan untuk upgrade bisa digantikan untuk menambah mesin dalam cluster, sehingga dari segi cost, biaya untuk upgrade server bisa ditekan. Dalam paper ini, indikator performa yang ingin didapatkan adalah *jumlah request yang bisa diselesaikan perdetik*

(request per second), waktu untuk menyelesaikan per satu request, dan throughput yang dihasilkan

2. LINUX VIRTUAL SERVER

Dengan makin berkembangnya internet maka server internet harus mampu menangani pertumbuhan kebutuhan yang pesat. Jumlah client yang harus disupport oleh server menngkat tajam, bahkan beberapa situs menerima ratusan bahkan ribuan koneksi client secara simultan.. Respon yang cepat dan availabilitas selama 24 jam setiap hari merupakan kebutuhan pokok untuk sebuah aplikasi bisnis yang berkompetisi untuk memberikan layanan akses terbaik. Tuntutan pada solusi hardware dan software yang dapat mensupport servis yang memiliki skalabilitas dan availabilitas yang tinggi dapat dikelompokkan sebagai berikut : [5]:

1. **Skalabilitas**, ketika beban pada service meningkat, sistem dapat dikembangkan dan disesuaikan dengan kebutuhan.
2. **Availabilitas** 24x7, secara keseluruhan harus dapat diakses (available) selama 24 jam sehari 7 hari seminggu, meskipun terjadi kerusakan hardware/software yg bersifat sementara.
3. **Manageability**, meskipun secara fisik sistem tersebut sangat besar, sistem tersebut harus mudah diatur.
4. **Keefektifan** biaya, keseluruhan sistem harus ekonomis untuk dikembangkan.

Sebuah single server biasanya tidak mampu menangani beban yang pertumbuhannya sangat pesat ini. Peng-upgrade-an server menimbulkan masalah yang sangat kompleks dan server itu sendiri merupakan sebuah titik kegagalan yang harus dijaga. Makin canggih server itu diupgrade maka makin tinggi biaya yang harus dibayar[5].

Cluster-cluster pada server yang dihubungkan oleh fast network muncul sebagai solusi arsitektur untuk membangun service yang memiliki tingkat availabilitas dan skalabilitas yang tinggi. Type arsitektur ini lebih terskala, lebih efektif biaya dan lebih reliable dibandingkan dengan sistem yang menggunakan multi prosesor. Bagaimanapun juga terdapat banyak tantangan untuk membuat fungsi peng-clusteran pada server ini berjalan secara efektif [5].

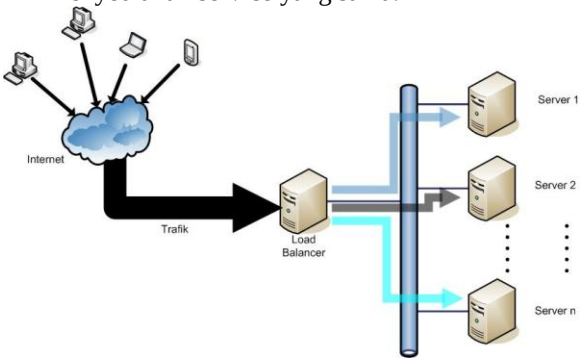
Linux Virtual server merupakan solusi untuk masalah diatas. Linux Virtual Server merupakan software yang langsung menghubungkan koneksi jaringan ke beberapa server yang saling membagi *workload*-nya, yang dapat digunakan untuk membangun servis yang memiliki skalabilitas dan availabilitas yang tinggi. Prototipe dai LVS telah digunakan untuk membangun banyak situs yang mempunyai beban yang sangat berat di internet, misalnya portal linux www.linux.com, soourceforge.net dan UK National JANET Web Chace service [5].

LVS menghubungkan koneksi jaringan ke beberapa server berdasarkan pada algoritma penjadwalan(*scheduling algorithm*) dan membuat pararel service terhadap cluster-cluster tersebut sehingga akan terlihat sebagai sebuah virtual servis pada sebuah IP Address single. Aplikasi client akan berhubungan dengan cluster tersebut seperti berhubungan dengan sebuah server single. Client tidak akan merasakan interaksi dengan cluster sehingga tidak perlu dimodifikasi. Skalabilitas diperoleh dengan menghapus dan menambahkan sebuah node pada cluster secara transparan. Availabilitas yang tinggi diperoleh dengan jalan mendeteksi kegagalan node atau daemon dan melakukan konfigurasi sistem secata tepat [5].

Virtual server merupakan suatu server dengan skalabilitas dan availabilitas tinggi yang dibangun dari sekumpulan server (*server cluster*). Arsitektur dari *cluster server* adalah transparan terhadap end-user sehingga end-user hanya akan melihat sebuah mesin tunggal saja [5].

Dalam prinsip kerjanya, *Virtual server* menerapkan *three-tier-architecture* (gambar 1) yang terdiri dari :

1. *Load balancer,* atau yang lazim pula disebut dengan LVS director yang merupakan mesin yang diletakkan di front-end dari keseluruhan sistem cluster. Load balancer ini bertugas untuk menyeimbangkan *request* dari client kepada kumpulan server.Load balancer adalah satu-satunya mesin yang terlihat oleh client yang akan mengakses ke server array[5].
2. *Server array*, merupakan kumpulan server yang menjalankan *network service* yang sama.dan dikonfigurasi untuk menjadi bagian dari LVS yang menerima hasil forward request dari client melalui load balancer [5].
3. *Shared Storage*, menyediakan *storage-space* bersama untuk kumpulan server, sehingga mudah bagi server untuk mengakses *content* dan menyediakan service yang sama.



Gambar 1. Arsitektur 3-tier virtual server

Load balancer merupakan *entry-point* tunggal dalam sistem *cluster*. *Load balancer* menggunakan algoritma *round-robin* untuk melakukan *context switching* pada *request*. Pada sisi *server array*, jumlah server bisa ditambahkan mengikuti beban

yang diterima oleh server.ketika semua server telah kelebihan beban, maka server baru seharusnya ditambahkan untuk menangani beban tambahan tersebut [5].

Shared storage merupakan tempat penyimpanan data yang bisa diakses oleh server secara berbagi. *Shared storage* digunakan agar kumpulan server bisa mengakses ke tempat data yang sama, karena service yang ditangani juga sama [5].

Load balancer menerima semua koneksi yang masuk dengan menggunakan teknik *IP Load Balancing*, kemudian memilih server dari server pool, menjaga status dari koneksi yg konkuren (*concurrent connection*) dan meneruskan paket, semua pekerjaan ini terjadi di dalam kernel seingga overhead dari load balancer ini rendah. Oleh karena itu Load balancer dapat menangani koneksi dalam jumlah yang besar dibandingkan dengan sebuah server biasa, karena load balancer dapat menjadwalkan server dalam jumlah yang banyak dan pada saatnya nanti dimungkinkan terjadi bottleneck pada sistem [5].

Node-node server pada arsitektur di atas dapat direplika baik untuk skalabilitas yang tinggi atau availabilitas yang tinggi. Skalabilitas didapat dengan cara menambah atau mengurangi jumlah server dalam cluster secara transparan. Ketika beban di sistem mencapai titik jenuh dan jumlah node-node server yang ada tidak mampu menangani, maka kita dapat menambahkan sejumlah node-node server untuk menangani beban kerja tersebut. Mengingat ketergantungan dari sebagian besar network services tidak terlalu tinggi, maka keseluruhan performa harus diukur secara linier dengan banyaknya node dalam sistem, sebelum load balancer menjadi sebuah bottleneck pada sistem. Karena server digunakan sebagai blok bangunan, maka performanya/rasio biaya dari keseluruhan sistem adalah sebesar jumlah server tersebut [5].

Salah satu keuntungan dari sistem yang ter-*cluster* (clustered system) adalah sistem ini memiliki hardware dan software yang redundan. Availabilitas yang tinggi dapat dicapai dengan mendeteksi kegagalan node atau daemon dan mengkonfigurasi sistem dengan tepat sehingga beban kerja dapat diambil alih oleh node-node yang lain di dalam cluster. Biasanya kita memiliki daemon untuk memonitor cluster yang dijalankan di load balancer yang digunakan untuk memonitor kondisi node-node server. Jika sebuah server tidak dapat dilakukan ICMP *ping* atau jika tidak ada respon dari servis selama periode waktu tertentu maka daemon tersebut akan menghilangkan atau men-*disable* server tersebut dari tabel penjadwalan load balancer, sehingga load balancer tidak akan menjadwalkan koneksi baru ke server yang rusak tersebut [5].

Fungsi *load balancer* pada system *virtual server* menggunakan algoritma tertentu untuk melakukan distribusi beban kepada *cluster server*, mekanisme ini disebut dengan *job scheduling*. Salah satu algoritma yang dipakai dalam sistem *Linux Virtual Server* adalah algoritma *round-robin* dan *weighted round-robin* [5].

Algoritma *round robin scheduling* merupakan algoritma yang didesain khusus untuk sistem *timesharing*. Terdapat satuan terkecil waktu yang disebut *time quantum*, atau *time slice*. *Time quantum* pada umumnya adalah 10 hingga 100ms. Antrian yang telah siap akan dimasukkan dalam antrian yang bentuknya *circular*. Untuk mengimplementasikan *round robin sheduling*, queue yang digunakan harus bersifat *First In First Out* [1].

Algoritma *weighted round-robin* adalah pengembangan lebih lanjut dari algoritma round-robin.Algoritma ini dapat mempertimbangkan beban server berdasarkan kapasitasnya. Tiap server diberi bobot sebuah nilai integer yang menggambarkan kapasitas prosesnya, nilai default untuk bobot ini adalah 1.

Pseudo code algoritma WRR adalah didefinisikan seperti gambar 2 sebagai berikut:

```
while (1) {
        i = (i + 1) mod n;
        if (i == 0) {
              cw = cw - 1;
              if (cw <= 0) {
                set cw the maximum weight
                of S;
                if (cw == 0) return NULL;
              }
            }
            if (W(Si) >= cw) return Si;
        }
```

G ambar 2. pseudo code WRR

Keterangan :

S = jumlah server (S_0,S_1,…S_{n-1})

i = indeks dari server terakhir yang terpilih dari

cw = current weight

(Di awal, variable i diberi nilai -1, dan cw diberi nilai 0)

Jika nilai $W(S_i)$=0 maka tidak ada server yang *available*, dan semua koneksi akan dilepas.

Pada algoritma ini semua server yang memiliki bobot lebih tinggi akan menerima koneksi lebih dulu dan akan mendapatkan koneksi lebih banyak dibandingkan dengan server yang mempunyai bobot lebih rendah, sedangkan untuk server-server yang memiliki bobot nilai yang sama akan mendapat distribusi koneksi baru yang sama juga. Misalnya, real server A,B,C memiliki bobot 4,3,2 maka penjadwalannya dapat mengikuti urutan berikut AABABCABC. WRR efisien digunakan untuk menjadwalkan *request* tetapi algoritma ini bisa menimbulkan ketidakseimbangan jika bobot tiap server sangat bervariasi [5].

Perbedaan *round-robin* dengan *weighted round-robin* adalah pada prioritas *job* yang ditangani. Dalam *weigted round-robin*, suatu server bisa diberi bobot

yang lebih tinggi dari server lainnya, sehingga *job scheduler* akan memprioritaskan job untuk server tersebut lebih dari server lainnya [5].

Dalam *cluster real server* pada LVS, kadangkala terdapat suatu ketidakseimbangan dalam hal penerimaan request, walaupun pada awalnya tujuan LVS adalah untuk menyeimbangkan hal tersebut.Ketidak seimbangan ini akan berakibat pada kinerja real server yang menurun pada suatu saat. Algoritma round-robin maupun weighted round-robin hanya melakukan pembagian berdasarkan nilai yang sudah ditentukan di awal. Jika pada saat sistem berjalan terjadi ketidak seimbangan, maka sulit bagi system administrator untuk melihat mesin manakah yang mengalami kelebihan beban. Kelemahan penggunaan *round-robin* dan *weighted round robin* pada LVS adalah pendistribusian beban tidak mempertimbangkan secara adaptif beban server pada saat itu.karena LVS tidak mengetahui jumlah request yang telah berhasil ditangani oleh server.

Dalam penelitian ini, akan dikembangkan suatu *job scheduler* yang adaptif terhadap beban dengan mengadopsi algoritma *round-robin*. Dengan pengembangan ini diharapkan *load balancer* akan bisa mendistribusikan beban kepada mesin yang tingkat keberhasilannya tinggi dalam menyelesaikan *request*.

Dalam LVS, terdapat 3 model *forwarding* oleh *load balancer*, yaitu *Network Address Translation, Tunneling,dan Direct Routing*. *Network Address Translation* merupakan suatu mekanisme yang melakukan *mapping* suatu alamat IP kepada alamat IP lain dengan tujuan transparansi komunikasi. *Tunneling* adalah suatu teknik yang meng-enkapsulasi IP datagram didalam IP datagram, sehingga memungkinkan datagram yang ditujukan untuk sebuah alamat IP bisa dibungkus dan diarahkan pada alamat IP yang lain. Sedangkan *Direct Routing* adalah mekanisme meneruskan suatu paket IP ke mesin server dengan mengganti header paket IP tersebut [5].

Penggunaan *Tunneling* dan *Direct Routing* membutuhkan server dengan kemampuan *tunneling* dan *direct routing*, sehingga tidak semua mesin dengan sistem operasi yang *multiplatform* bisa dijadikan anggota *cluster*. Penelitian ini nantinya akan menggunakan metode NAT sebagai model *forwarding*, karena sifat NAT yang melakukan mapping dari *load balancer* menuju *real server*, sehingga memudahkan implementasi pada sistem yang *multiplatform* [5].

3. APACHE BENCHMARK

Apache benchmark (AB) merupakan software yang dikembangkan oleh Apache Foundation sebagai alat untuk melakukan benchmark kepada sebuah Web server, khususnya Apache Web Server.

Apache Benchmark melakukan benchmarking terhadap suatu web server dengan cara memberikan request dengan parameter-paramater tertentu. parameter-parameter penting yang bisa dijadikan kunci benchmarking adalah jumlah request dan jumlah konkurensi. Beberapa nilai yang bisa diganti untuk menyesuaikan beban yang akan diberikan pada server adalah sebagai berikut :

1. Jumlah request yang dilakukan
   Jumlah request bisa diubah-ubah sesuai dengan keinginan user, semakin banyak jumlah request yang diberikan maka server yang diberi request akan semakin terbebani.
2. Konkurensi
   Konkurensi adalah request bersamaan yang akan dilakukan oleh ab kepada server.konkurensi ini nilai maksimumnya adalah 1000, jika lebih dari nilai tersebut, ab akan mengeluarkan pesan error karena terlalu banyak file yang terbuka.
3. Timelimit
   Waktu maksimum apache benchmark untuk menunggu koneksi selesai dari server.
4. Postfile
   Selain bisa memberikan jumlah request dan konkurensi, ab juga emungkinkan untuk diberikan inputan simulasi yang berupa file. Inputan tersebut nantinya akan bisa diset oleh pengguna apakah akan menggunakan metode POST ataukah metode GET .

4. IPVSADM

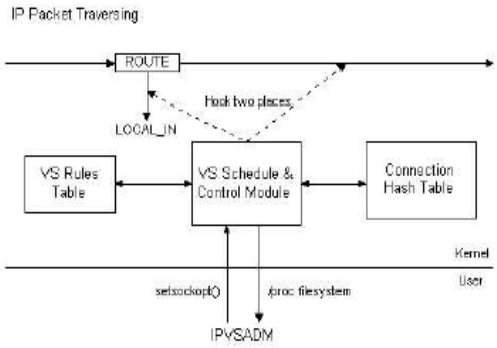
IPVSADM merupakan tool yang disediakan oleh Linux virtual server, IPVSADM merupakan user space program yang digunakan oleh administrator LVS untuk melakukan administrasi server LVS. IPVSADM menggunakan setsockopt untuk merubah nilai nilai VS rules di dalam kernel, dan membaca rules dari /proc file system.

Dalam implementasi sistem LVS, seperti gambar 4.1, "VS Schedule & Control Module" merupakan modul utama dari LVS. Modul tersebut melakukan hook pada dua tempat pada lokasi pemrosesan paket IP di dalam kernel untuk melakukan proses penulisan ulang paket IP yang akan digunakan untuk load balancing.

Untuk koneksi baru, modul tersebut akan mencari VS rules hash table, dan kemudian akan melakukan pemeriksaan pada Connection Hash Table untuk koneksi yang sudah ada.

Hash table pada LVS dirancang untuk bisa menampung jutaan hash table koneksi yang konkuren. Dan tiap-tiap koneksi hanya menghabiskan tempat 128 byte dalam memory Load balancer.

LVS menerapkan ICMP handling untuk virtual services. Paket ICMP yang datang untuk virtual services akan dimodifikasi dan diteruskan kepada server yang dituju.



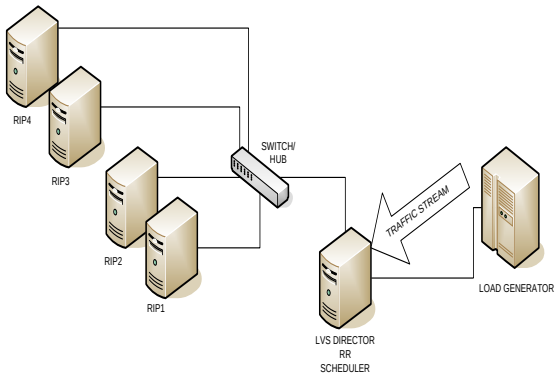
Gambar 4.1 Struktur internal LVS

5. KINERJA LINUX VIRTUAL SERVER

Pada penelitian ini, dilakukan ujicoba dengan konfigurasi LVS sebagai berikut, terdapat 3 bagian dalam sistem ujicoba ini yaitu server cluster yang berupa 4 buah PC ,Intel Pentium 133Mhz, RAM 64MB HDD 3GB, dengan Sistem Operasi Linux Mandrake 7.2 Kernel 2.2.20, menjalankan service Apache HTTPD 2 sebagai web server, kemudian LVS Director yang berupa 1 PC, Intel PII, 450 Mhz, RAM 128M HDD 4GB menjalankan sistem operasi Linux Debian Kernel 2.6.

Sedangkan untuk mesin client yang melakukan simulasi trafik untuk membebani server adalah

berupa 1PC, Intel P4, 2.4Ghz, RAM 512M, HDD 80GB, yang menjalankan Sistem Operasi Linux Debian ,Kernel 2.6, yang menjalankan software Apache Benchmark. Untuk lebih jelasnya bisa melihat pada gambar 5.1 tentang konfigurasi jaringan ujicoba.



Gambar 5.1Konfigurasi jaringan ujicoba

6. SKENARIO UJICOBA

Dalam penelitian ini, kinerja adalah indikator penting. Sehingga terdapat 3 aspek kinerja yang akan diukur dari sisi client, yaitu mendapatkan nilai throughput, kemudian jumlah request per second, dan waktu per request. Ujicoba ini nantinya akan melakukan pengukuran dalam 3 jenis request yaitu, 1000 request, 3000 request, dan 5000 request. Ujicoba ini dilakukan dengan membandingkan dua jenis konfigurasi, yang pertama adalah menggunakan single server yang mana test dilakukan hanya pada sebuah server saja tanpa LVS (nantinya dalam hasil ujicoba disebut dengan single). Dan kemudian konfigurasi yang menggunakan LVS (disebut LVS). Keduanya nanti akan dibandingkan hasilnya. Hasil yang akan disimpulkan merupakan rata-rata dari perhitungan request 1000,3000 dan 5000. Ujicoba dilakukan mengggunakan LVS yang mengimplementasikan scheduler round-robin.

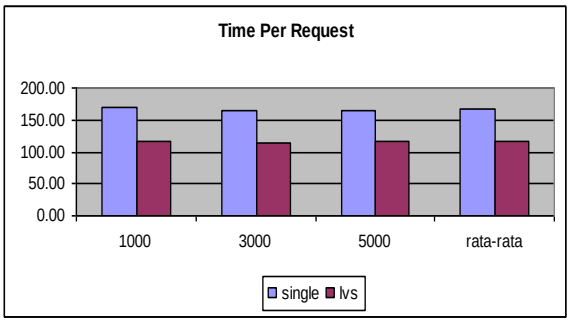
7. HASIL UJICOBA

Setelah ujicoba dilakukan, maka didapatkan hasil-hasil berupa nilai-nilai indikator yang telah disebutkan, yaitu:

Time per request adalah waktu yang dibutuhkan untuk mendapatkan response dalam satu request yang dikirimkan. Dalam perbandingan ini, dapat dilihat bahwa terjadi penurunan waktu per request sekitar 43.65 persen jika sistem menggunakan LVS (Tabel 3.1 dan Gambar 3.3) , sedangkan untuk penggunaan IPv6 terjadi penurunan sekitar 70 persen jika sistem menggunakan LVS (Tabel 3.2 dan Gambar 3.4).

Tabel 7.1. hasil uji coba *time per request*

time per request	#req	single	LVS	(+/-)
	1000	170.53	117.20	-45.50
	3000	165.55	114.90	-44.08
	5000	164.28	116.20	-41.37
satuan ms	Rata-rata	166.78	116.10	-43.65

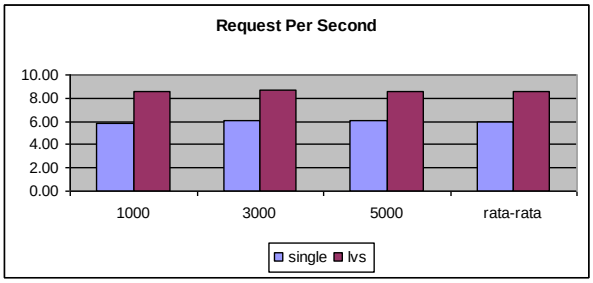


Gambar 7.1. Grafik hasil uji coba time per request

Request per second adalah jumlah request yang bisa ditangani server dalam satu detik. Pada penggunaan LVS di IPv4 dibandingkan individual server terdapat peningkatan sekitar 43.60 persen (Tabel 3.3 dan Gambar 3.5), sedangkan penggunaan pada IPv6 LVS bisa meningkatkan sampai 70.86 persen (Tabel 3.4 dan Gambar 3.6), berikut adalah tabel dan grafiknya.

Tabel 7.2 Tabel hasil uji coba *request per second*

#req per second	#req	single	LVS	(+/-)
	1000	5.86	8.53	45.50
	3000	6.04	8.70	44.04
	5000	6.09	8.60	41.27
satuan jumlah	rata-rata	6.00	8.61	43.60



Gambar 7.2. Grafik hasil uji coba request per second pada IPv4

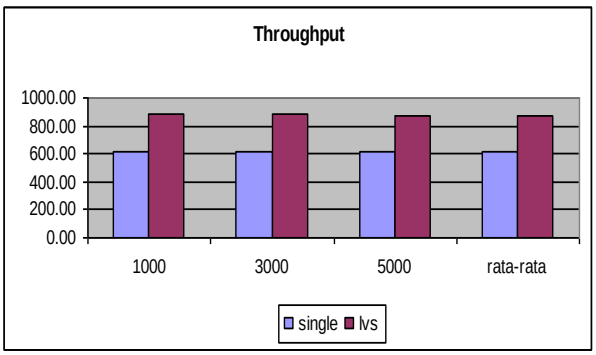
Tabel 7.4 Hasil perbandingan keseluruhan

request	Single			LVS			(+/-)		
	Rps #req	Tpr ms	Throughput KB/s	Rps #req	Tpr ms	Throughput KB/s	Rps #req	Tpr ms	Throughput KB/s
1000	5.863	170.525	612.825	8.530	117.200	879.900	45.501	-31.271	43.581
3000	6.040	165.550	618.525	8.700	114.900	880.300	44.040	-30.595	42.322
5000	6.088	164.275	617.850	8.600	116.200	866.900	41.273	-29.265	40.309
	5.997	166.783	616.400	8.610	116.100	875.700	43.605	-30.377	42.071

Throughput adalah banyaknya data (dalam satuan Kilobyte) yang diterima per detik. Penggunaan LVS bisa meningkatkan throughput sekitar 42.07 persen (Tabel 3.5 dan Gambar 3.7), sedangkan untuk IPv6 bisa meningkatkan sekitar 67.28 persen (Tabel 3.6 dan Gambar 3.8).

Tabel 7.3.
Tabel hasil uji coba perbandingan *throughput*

throughput	#req	single	LVS	(+/-)
	1000	612.83	879.90	43.58
	3000	618.53	880.30	42.32
	5000	617.85	866.90	40.31
satuan KB/s	rata-rata	616.40	875.70	42.07



Gambar 7.3. Grafik hasil uji coba perbandingan throughput

Dari data-data yang diperoleh diatas, dibuat tabel 7.4 yang membandingkan hasil uji coba secara keseluruhan. Dari tabel 7.4 tersebut diperlihatkan jika dibandingkan antara penggunaan single server dibandingkan dengan penggunaan LVS, ternyata penggunaan LVS ternyata meningkatkan performa dalam melayani jumlah request yang ditangani perdetik (*Request Per Second*), yaitu meningkat secara rata-rata sekitar 43 persen. *Time per request* (Waktu yang dibutuhkan untuk satu request) juga rata-rata menurun sebesar 30.377 persen. *Throughput* (Data yang diterima per detik) juga rata-rata meningkat 42 persen.

8. KESIMPULAN

Penggunaan Linux Virtual Server yang menggunakan round-robin scheduler dalam lingkungan percobaan yang telah disebutkan, ternyata meningkatkan performa dari sebuah sistem yang hanya menggunakan single server, dari hasil penelitian diatas didapatkan bahwa rata-rata kenaikan performa adalah pada rata-rata Jumlah request persecond sekitar 43.605 persen, kemudian penurunan waktu untuk satu request rata-rata 30 persen, dan kenaikan throughput rata-rata 42 persen.

9. DAFTAR PUSTAKA

1. Avi Silberschatz, Peter, Applied Operating System Concepts, John Wiley & Sons, 2000.
2. Forouzan, Behrouz, TCP/IP Protocol Suite, McGraw-Hill, 2000.

3. Hemant M. Chaskar, Upamanyu Madhow, Fair scheduling with tunable latency: a round-robin approach,IEEE/ACM Transactions on Networking Volume 11 Issue 4, 2003..
4. James Aweya, Michel Ouellette, Delfin Y. Montuno, Bernard Doray, Kent Felske, "An adaptive load balancing scheme for web servers" , International Journal of Network Management volume 12 issue 1, 2002.
5. LVS, Linux virtual server, http://www.linuxvirtualserver.org,2004,
6. Peterson, Larry L., Davie, Bruce S., Computer networks A System Approach, Morgan-Kauffman,1999.
7. Putrycz, Erik, "Design and implementation of a portable and adaptable load balancing framework", Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research. 2003,
8. R. Gilligan, S. Thomson, J. Bound, W. Stevens,RFC 2133- "Basic Socket Interface Extensions for IPv6", 1997.
9. [9] S. Deering, R. Hinden, RFC 1883 - Internet Protocol, Version 6 (IPv6) Specification,1995.
10. [S. Deering, R. Hinden, "RFC 3513- Internet Protocol", Addressing Architecture,2003.
11. USAGI, USAGI project overview, http://www.linux-ipv6.org/overview.html#overview.