JIMU: Jurnal Ilmiah Multi Disiplin
Vol. 03, N0. 01, Tahun 2025, Hal. 397 - 413, ISSN: 3031-9498 (Online)

Konfigurasi *Load Balancing* Pada Server Dengan Menggunakan Algoritma Round Robin di Universitas Negeri Makassar

Khairul Imam Muhammad Jufri[1], Mustari S. Lamada[2], Muhammad Agung[3]
[1,2,3] Universitas Negeri Makassar
Email: khairulimam2000@gmail.com

Article Info

Article history:

Receved October 26, 2024
Revised November 15, 2024
Accepted November 20, 2024

Keywords:

Server, Load Balancer, Weighted Round Robin, Makassar State University

ABSTRACT

The development of information technology, including the Internet, is rapid and is used globally to find and exchange information. Many government agencies, such as Makassar State University, are attempting to improve information technology. However, the so-called information technology must have shortcomings because almost 54,461 students access the same website, which causes the website to experience downtime. The solution applied is a load balancing system with the Weighted Round Robin algorithm, which divides the server load based on a certain scale to optimize performance and prevent server downs. This research aims to: (1) analyze the comparison between refresh and login, (2) compare server performance when receiving 100, 500, and 1000 requests using Apache Bench, (3) measure the number of bytes when accessing tools such as calendars, and (4) compare servers with and without load balancing. The results show that the bytes generated at login were greater than those generated at refresh. In addition, the more requests received, the longer is the server processing time. For calendar access, 1,336 sessions were recorded in 2 minutes 45 seconds. In conclusion, servers that use load balancers are more efficient and faster than servers that do not use load balancing.

Article Info

Article history:

Receved October 26, 2024
Revised November 15, 2024
Accepted November 20, 2024

Keywords:

Server, Load Balancer, Weighted Round Robin, Universitas Negeri Makassar

ABSTRACT

Perkembangan teknologi informasi, termasuk internet, sangat cepat dan digunakan secara global untuk mencari serta bertukar informasi. Banyak instansi pemerintahan, seperti Universitas Negeri Makassar, berupaya meningkatkan teknologi informasinya. Namun yang dinamakan teknologi informasi pasti memiliki kekurangan, dikarenakan hampir sekitar 54.461 mahasiswa mengakses website yang sama yang menyebabkan website yang dimiliki mengalami *downtime*. Solusi yang diterapkan adalah sistem *load balancing* dengan algoritma *Weighted Round Robin*, yang membagi beban server berdasarkan skala tertentu untuk mengoptimalkan kinerja dan mencegah *downserver*. Penelitian ini bertujuan untuk : (1) menganalisis perbandingan antara *refresh* dan *login*, (2) membandingkan performa server saat menerima 100, 500, dan 1000 permintaan menggunakan *Apache Bench*, (3) mengukur jumlah *bytes* saat mengakses alat seperti kalender, dan (4) membandingkan server dengan dan tanpa *load balancing*. Hasil penelitian menunjukkan bahwa *bytes* yang dihasilkan saat *login* lebih besar dibanding *refresh*. Selain itu, semakin banyak permintaan yang diterima, semakin lama waktu pemrosesan server. Untuk akses kalender, tercatat 1.336 sesi

dalam 2 menit 45 detik. Kesimpulannya, server yang menggunakan *load balancer* lebih efisien dan cepat dibandingkan server tanpa *load balancing*.

Corresponding Author:

Nama penulis: Khairul Imam Muhammad Jufri
Universitas Negeri Makassar
Email: khairulimam2000@gmail.com

Pendahuluan

Perkembangan teknologi informasi yang pesat, termasuk internet, telah memungkinkan akses informasi di mana saja dan kapan saja. Seiring kemajuan ini, instansi pemerintah dan swasta berlomba-lomba mengadopsi teknologi informasi yang canggih untuk meningkatkan efisiensi pengelolaan data (Azhar et al., 2022). Namun, peningkatan jumlah pengguna sering kali menyebabkan beban server berlebih, waktu respons lambat, hingga kegagalan sistem. Salah satu solusi untuk mengatasi tantangan ini adalah *load balancing*, sebuah teknologi yang mendistribusikan beban server secara merata untuk meningkatkan kinerja dan ketersediaan layanan. Salah satu algoritma yang sering digunakan adalah *Weighted Round Robin* (WRR), yang membagi beban berdasarkan kapasitas server sehingga server berkinerja tinggi dapat menangani lebih banyak permintaan (Hanafiah, 2021).

Universitas Negeri Makassar (UNM) merupakan salah satu institusi yang menggunakan teknologi informasi dalam pembelajaran, seperti *Learning Management System* (LMS) SYAM-OK, untuk mendukung pembelajaran daring dan *blended learning* (Agung, 2022). Namun, dengan jumlah mahasiswa yang mencapai 54.461 orang, seringkali terjadi *downtime* pada LMS SYAM-OK, terutama pada masa-masa ujian (Samad, 2021). Hal ini disebabkan oleh tingginya akses serentak yang melebihi kapasitas server. Untuk mengatasi masalah ini, penerapan *load balancing* berbasis algoritma *Weighted Round Robin* (WRR) diharapkan dapat meningkatkan kinerja server dan mencegah *downtime* yang terjadi (Baharuddin et al., 2023).

Beberapa penelitian sebelumnya telah mengeksplorasi penerapan *load balancing* untuk mengatasi masalah serupa. Hakim dkk (2019) melakukan pengujian *load balancing* pada web server menggunakan *NGINX*, dengan tujuan untuk mendistribusikan beban trafik pada aplikasi Sistem Informasi Pelayanan Desa di Kabupaten Purworejo. Penelitian ini menggunakan metode kuantitatif dan kualitatif untuk menguji kinerja masing-masing server dan membuktikan bahwa penggunaan Ubuntu sebagai server *load balancing* berhasil mencapai kecepatan akses yang stabil meskipun banyak pengguna yang mengakses secara bersamaan (Hakim et al., 2019).

Penelitian lain oleh Hidayat dkk (2021) mengimplementasikan metode *Per Connection Classifier* (PCC) dan *failover load balancing* pada Balai Besar Pelatihan Kesehatan (BBPK) Jakarta. Penelitian ini bertujuan untuk mendistribusikan beban trafik antar dua ISP, dengan fokus pada pencegahan kegagalan koneksi jika salah satu ISP mengalami gangguan. Metode

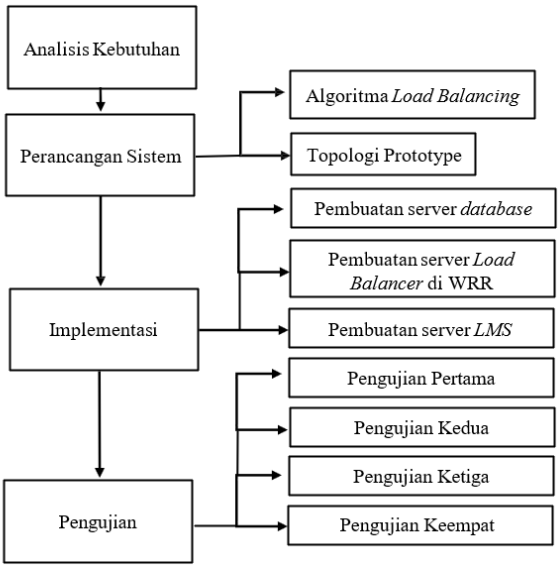
deskriptif digunakan dalam penelitian ini, yang menunjukkan bahwa penerapan *load balancing* berbasis PCC mampu meningkatkan keandalan dan ketersediaan sistem secara signifikan (A. S. Hidayat et al., 2021).

Hidayat (2022) mengembangkan prototipe sistem *load balancing* menggunakan algoritma *Least Connection* pada server di Sekolah Vokasi. Penelitian ini membandingkan kinerja server dengan dan tanpa *load balancing* dalam tiga skenario: akses halaman web LMS, server dalam kondisi hidup dan mati, serta kinerja server dengan algoritma *Least Connection*. Meskipun berbeda algoritma dengan penelitian ini, penelitian ini tetap menunjukkan pentingnya penggunaan *load balancing* dalam meningkatkan performa server (M. . Hidayat, 2022).

Berdasarkan dari beberapa penelitian sebelumnya serta masalah yang beredar, disulkan penelitian berikut yang bertujuan untuk merancang dan menganalisis penerapan *load balancing* berbasis algoritma *Weighted Round Robin* (WRR) pada server UNM. Dengan pendekatan ini, diharapkan server dapat mendistribusikan beban secara optimal, mencegah *downtime*, dan meningkatkan pengalaman pengguna dalam mengakses LMS SYAM-OK. Penelitian ini akan menggunakan simulasi berbasis *virtual machine* untuk menguji kinerja server sebelum dan sesudah penerapan *load balancing*.

Metode

Metode penelitian yang digunakan pada penelitian ini yaitu metode eksperimental. Berikut adalah diagram proses atau tahapan-tahapan dari penelitian, diantaranya :



Gambar 1. Diagram Tahapan - Tahapan Penelitian

Pada Gambar 1 Menunjukkan tahapan - tahapan dari penilitian yang akan dilakukan, diantaranya :

2.1 Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk memastikan bahwa semua elemen yang diperlukan untuk membangun sistem *load balancing* telah dipahami dan disiapkan. Langkah ini

melibatkan pengumpulan informasi melalui wawancara dengan staf terkait dan tinjauan literatur pada jurnal-jurnal yang relevan tentang *load balancing*. Hasil dari analisis ini digunakan untuk merancang sistem yang optimal.

2.2 Perancangan Sistem

Perancangan sistem dilakukan dengan membangun *prototype* menggunakan algoritma *Weighted Round Robin* (WRR) untuk *load balancing*. WRR membagi beban antar server berdasarkan bobot tertentu. Topologi sistem dirancang menggunakan empat unit *virtual machine*, yang masing-masing berfungsi sebagai *load balancer*, *web server*, dan *database server*. *Web server* bertugas melayani permintaan pengguna melalui protokol *HTTP* atau *HTTPS* dan mengirimkan respon berupa dokumen *HTML* atau halaman web. Sementara itu, *database server* digunakan untuk menyimpan dan mengelola data secara terstruktur, sehingga memudahkan pengguna dalam pencarian, penyimpanan, dan penghapusan informasi.

2.3 Implementasi

Implementasi melibatkan pembuatan *prototype load balancing* secara virtual menggunakan *VirtualBox*. Tahapan implementasi meliputi,

- **Pembuatan *Load Balancer*** : Mengonfigurasi *haproxy* dengan algoritma WRR untuk mendistribusikan beban.
- **Pembuatan *Web Server*** : Menggunakan *Apache* sebagai *web server* dan *Moodle* sebagai LMS.
- **Pembuatan *Database Server*** : Menggunakan *MariaDB* sebagai database management system untuk mendukung *Moodle*.

2.3 Pengujian

Pengujian adalah tahap *prototype* sistem teknik *Load Balancing* diuji dengan empat macam pengujian dan hasil dari pengujian tersebut dilihat melalui halaman statistik dari *haproxy*. Pada pengujian pertama dilakukan perbandingan antara *refresh* dan *login* pada halaman web melalui browser. Pengujian kedua dilakukan dengan melakukan pengaksesan salah satu fitur yang ada pada halaman web. Pengujian ketiga dilakukan perbandingan dengan mengirim 100, 500 dan 1000 *request* melalui aplikasi *apache bench*. Pengujian keempat dilakukan dengan membandingkan hasil *apache bench* dari pengiriman *request* untuk *web server* tanpa *Load Balancing* dengan *web server* yang menggunakan *Load Balancing*.

Hasil

3.1 Analisis Kebutuhan

Pada tahap analisis kebutuhan dilakukan pencatatan kebutuhan apa aja yang dibutuhkan dalam pembuatan *prototype* sistem *load balancing*. Aplikasi yang digunakan untuk teknik *load balancing* adalah *haproxy* versi 2.3. *Haproxy* dipilih karena pada *haproxy* bersifat gratis atau *open source* dan dapat berjalan pada sistem operasi *linux*. Pada *haproxy* juga dapat meminimalisir terjadinya *downtime* dengan menggunakan teknik *load balancing* yang dapat membagi beban *request* ke beberapa server. *Haproxy* menyediakan fitur untuk melihat halaman statistik dari akses user ke *web server* untuk kebutuhan pemantauan server web.

Aplikasi LMS yang digunakan sebagai keluaran *web server* adalah LMS *moodle* versi 3.8. *Moodle* dipilih karena LMS ini bersifat gratis atau *open source*, mudah digunakan, dapat

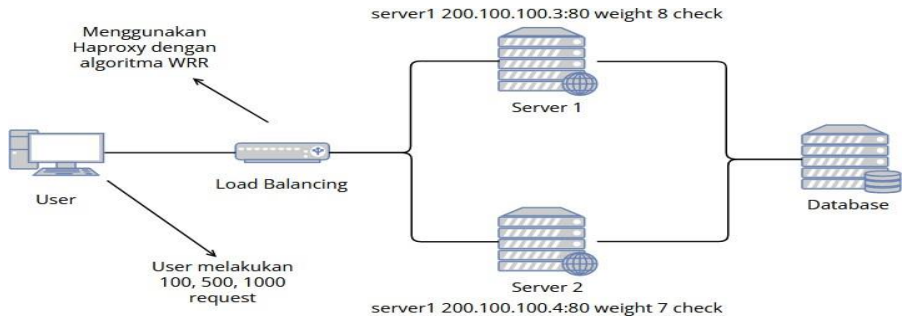
mendukung pengajaran dan pembelajaran secara online, dan dapat diterapkan pada sistem operasi *linux*. Aplikasi yang digunakan untuk mengatur database adalah *mariadb* versi 10.1. *Mariadb* dipilih karena bersifat gratis atau *open source*, dapat digunakan pada sistem operasi *linux* dan dapat digunakan pada *moodle* versi 3.8.

Server *moodle* untuk bekerja secara optimal dalam menangani *request* yang berjalan bersamaan (*concurrent user*) menggunakan aturan umum *concurrent user* = RAM(GB)*50 atau 1GB RAM untuk 50 concurrent. Pada prototype virtual menggunakan dua unit server virtual dengan masing-masing RAM sebesar 512MB yang dapat menangani *request* yang berjalan bersamaan sekitar 50 *concurrent user*.

3.2 Perancangan Sistem

Pada tahap perancangan sistem melibatkan pembuatan *prototype* dengan algoritma *Weighted Round Robin* (WRR) untuk *load balancing* dan desain topologi jaringan. Algoritma WRR, pengembangan dari *Round Robin*, membagi beban berdasarkan kapasitas server dengan menetapkan bobot manual pada setiap server. Server dengan kapasitas lebih besar diberi prioritas lebih tinggi dalam menangani job.

Perancangan topologi *prototype* sistem dibangun menggunakan empat unit *virtual machine* ubuntu server 18.04 yang dibagi menjadi 3 jenis perangkat. Dua unit server sebagai *web server* dengan *apache* sebagai aplikasi dari *web server*. Satu unit server sebagai server database untuk menyimpan data dari LMS dengan menggunakan *mariadb* sebagai aplikasi management database. Satu unit server sebagai *load balancer* untuk sistem *load balancing* menggunakan aplikasi *haproxy* dengan algoritma WRR sebagai algoritma pengatur jalur *request*. Adapun topologi load balancing dapat dilhat pada gambar 2.



Gambar 2. Topologi Sederhana *load balancing*

Adapun bobot skala yang diterapkan adalah 8:7, dengan server1 memiliki bobot 8 dan server2 memiliki bobot 7. Penentuan bobot ini didasarkan pada perbedaan kapasitas masing-masing server, menggunakan rumus yang ditampilkan pada Gambar 2. Sebagai contoh, CPU1 memiliki kapasitas kinerja 50% dan memori1 30%, sementara CPU2 memiliki kapasitas kinerja 40% dan memori2 30%. Rumus tersebut terdiri dari dua tahap, yaitu menggabungkan setiap pasangan bobot dan menormalisasikannya. Data dihitung dengan :

Weighted_pair_1 = CPU1 + Memori1 = 50% + 30% = 80%

Weighted_pair_2 = CPU2 + Memori2 = 40% + 30% = 70%

Dengan bobot 8:7, total bobot dihitung :

Total_weighted = 80% + 70% = 150%.

Kemudian, dilakukan normalisasi:

Weighted_pair1_norm = 80% / 150% = 0,53333

Weighted_pair2_norm = 70% / 150% = 0,46667

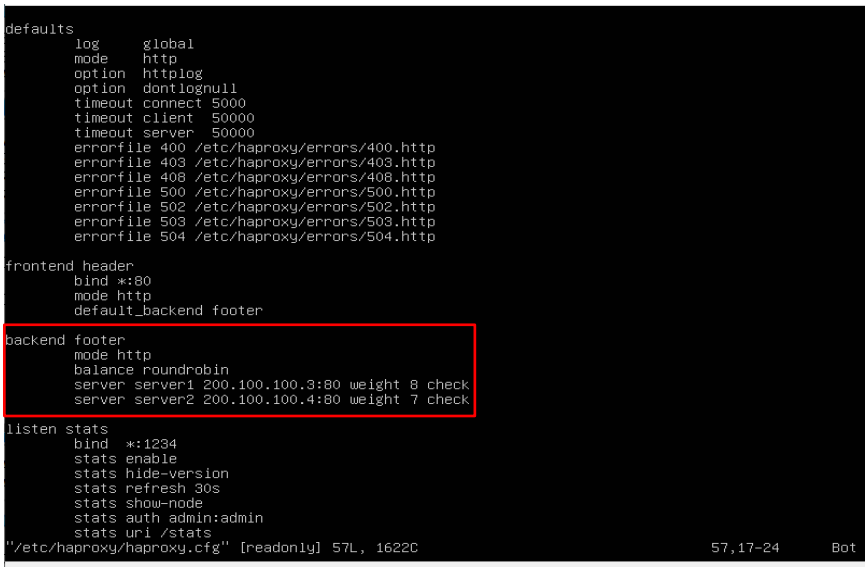
Hasilnya menunjukkan bahwa server1 dengan bobot 8 menghasilkan kinerja 0,53333 dari total, sedangkan server2 dengan bobot 7 menghasilkan kinerja 0,46667 dari total.

3.3 Implementasi

Pada tahap implementasi membuat konfigurasi *prototype* dengan *software Virtual Box* dari topologi yang sudah dibuat dan dirancang. Pembuatan penelitian *prototype* ini dibagi menjadi 3 bagian, yaitu pembuatan server *database*, pembuatan sistem *load balancer* dan pembuatan *web* LMS SYAM-OK. Konfigurasi awal dalam pembuatan setiap server adalah pemberian alamat IP. Pemberian alamat IP dapat dilihat pada gambar 3.

Gambar 3. Pemberian Alamat IP

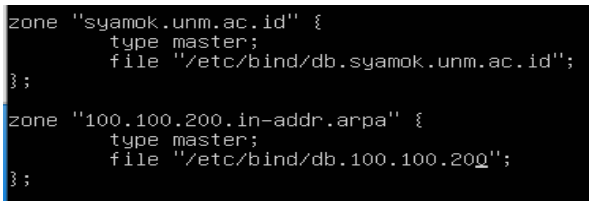
Pada pemberian IP dilakukan pada *network* yang sama, hal tersebut berguna untuk memudahkan admin dalam mengkonfigurasi. Pemberian IP juga berguna sebagai identitas setiap *device*, seperti server *load balancer* diberikan IP 200.100.100.1/24 dan 200.100.100.2/24. Selanjutnya setelah memberikan alamat IP ke setiap server, saatnya melakukan konfigurasi *load balancing* di server *load balancer*. Konfigurasi *load balancing* dapat dilihat pada Gambar 4.



Gambar 4. Konfigurasi *haproxy.cfg*

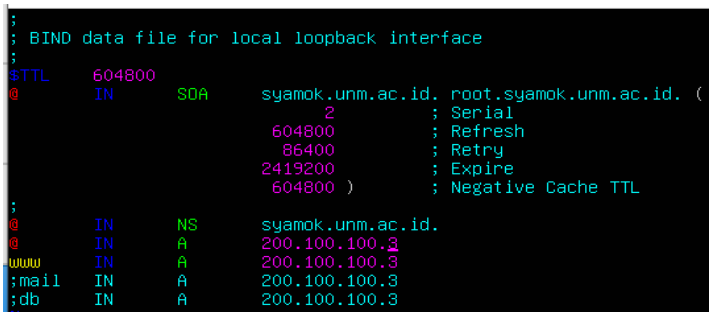
Pembuatan konfigurasi pada server *load balancer* dilakukan menggunakan haproxy. Konfigurasi *load balancing* dilakukan dengan menginstall *haproxy* terlebih dahulu agar file

haproxy dapat muncul. Pembuatan file terdapat pada *directory /etc/haproxy*. File ini digunakan untuk menentukan algoritma *load balancing* (*round robin*), mengatur halaman statistik *haproxy*, dan membagi jalur akses antar alamat IP server web. Selanjutnya melakukan konfigurasi DNS pada *direktori /etc/bind.*



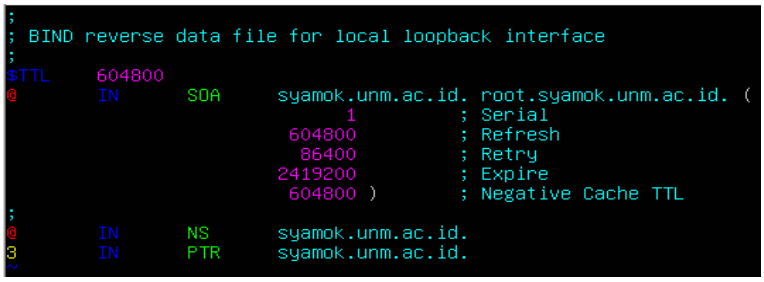
Gambar 5. Konfigurasi *named.conf.default-zones*

Pada gambar 5, dilakukan pembuatan DNS dengan mendefinisikan zona *domain server* pada *file* named.conf.local. Pada *file* ini, alamat atau *domain web* dimasukkan pada bagian *zone* yang berisi lokasi program yang diakses *domain* tersebut. Adapun zona yang dibuat pada *file* ini adalah zona untuk *file forward* dan *file reverse*. Setelah melakukan konfigurasi di *file zone*, kemudian dilakukan konfigurasi di *file forward*.



Gambar 6. Konfigurasi *db.syamok.unm.ac.id*

Pada gambar 6 menunjukkan pembuatan *file forward* untuk mengonversi DNS menjadi alamat IP, yaitu *db.syamok.unm.ac.id.* File ini mencakup nama *domain syamok.unm.ac.id* dengan tipe A *record*, yang mengarahkan *domain* tersebut ke alamat IP 200.100.100.3. Kemudian terdapat pembuatan *file reverse* untuk konversi alamat IP ke domain dengan nama *db.192*.



Gambar 7. Konfigurasi db.100.100.200

Pada gambar 7 terdapat file berisi nama *domain* yang digunakan yaitu *syamok.unm.ac.id* dan tipe *record* yang digunakan yaitu PTR *record* yang berfungsi untuk mengarahkan alamat IP 200.100.100.3 ke *domain syamok.unm.ac.id*. Langkah berikutnya penambahan nama *domain* dan mengubah *nameserver* pada *file* /etc/resolv.conf menjadi alamat IP pada *server* seperti pada

gambar 8, Penambahan *script* tersebut bertujuan untuk mendefinisikan nama *domain* dan *nameserver* pada *server* tersebut.

```
domain syamok.unm.ac.id
search syamok.unm.ac.id
nameserver 200.100.100.1
nameserver 8.8.8.8
nameserver 127.0.0.53
```

Gambar 8. Konfigurasi *resolv.conf*

Selanjutnya dilakukan Pembuatan *server database* dilakukan dengan menggunakan *database mariadb* dengan alamat IP 200.100.100.5. Pada database untuk dapat digunakan oleh LMS *moodle* diperlukan tambahan konfigurasi pada bagian *basic settings* pada file *50-server.cnf* pada *direktori /etc/mysql/mariadb.conf*. *Script* yang ditambahkan adalah jenis *storage engine* yang digunakan yaitu *innodb* dan tipe format file *innodb* yaitu *barracuda* seperti yang ditunjukkan pada gambar 9.

```
#
# See the examples of server my.cnf files in /usr/share/mysql/
#

# this is read by the standalone daemon and embedded servers
[server]

# this is only for the mysqld standalone daemon
[mysqld]

#
# * Basic Settings
#
user            = mysql
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
port            = 3306
basedir         = /usr
datadir         = /var/lib/mysql
tmpdir          = /tmp
lc-messages-dir = /usr/share/mysql
skip-external-locking

default_storage_engine = innodb
innodb_file_per_table = 1
innodb_file_format = Barracuda
innodb_large_prefix = 1

# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address            = 200.100.100.5
```

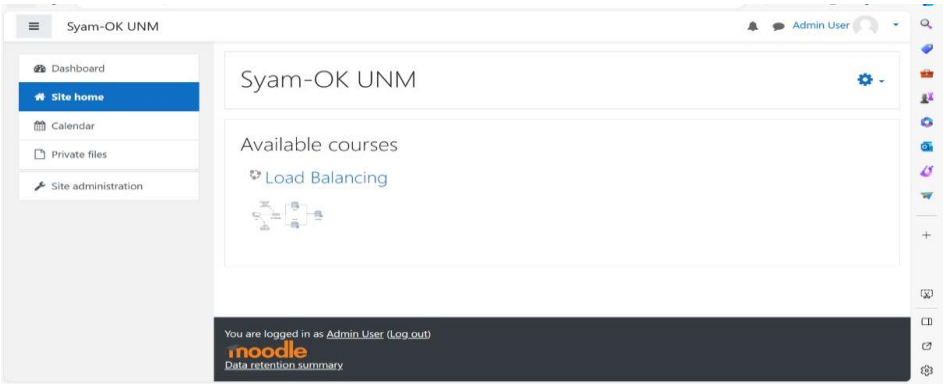
Gambar 9. Konfigurasi *50-server.cnf*

Langkah selanjutnya adalah membuat database bernama *moodle* pada terminal MySQL di Linux. Selanjutnya, dibuat pengguna *usrmoodle* dengan lokasi database di alamat IP 200.100.100.5 agar dapat diakses oleh *server* lain dalam jaringan. Hak akses penuh diberikan kepada *usrmoodle* menggunakan perintah *grant all privileges*, diikuti oleh *flush privileges* untuk melakukan *reload* pada database dan memastikan perubahan tersimpan serta berfungsi dengan baik seperti pada gambar 10.

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 30
Server version: 10.1.47-MariaDB-0ubuntu0.18.04.1 Ubuntu 18.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE moodle DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> CREATE USER 'usrmoodle'@'200.100.100.5  IDENTIFIED BY 'secret';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON moodle.* TO 'usrmoodle'@ 200.100.100.5';
Query OK, 0 rows affected (0.02 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)
```

Gambar 10. Membuat *database moodle* pada *mysql*

Selanjutnya, dibuat *Server* LMS *Moddle* menggunakan *web server Apache* dengan IP 200.100.100.3 untuk *server* 1 dan 200.100.100.4 untuk *server* 2. Proses dimulai dengan mengunduh *Moodle* versi 3.8 dari situs resminya dan mengekstraknya pada *server* LMS. Kemudian folder data dan web dibuat pada direktori */var/www/html/moodle* untuk menyimpan data *Moodle*. Selanjutnya, dibuat file konfigurasi *virtual host* bernama *moodle.syamok.unm.ac.id.conf* di direktori */etc/apache2/sites-available*, yang berisi nama *server syamok.unm.ac.id* dan direktori */var/www/html/moodle/web* seperti ditampilkan pada gambar 11. File *virtual host* diaktifkan dengan perintah *a2ensite syamok.unm.ac.id.conf*, kemudian instalasi dilanjutkan melalui browser.



Gambar 11. Tampilan web LMS setelah *login*

3.4 Pengujian

Pengujian dilakukan dengan mengakses *web* melalui *browser* dengan menggunakan alamat *domain syamok.unm.ac.id* dan hasil pengujian dicek melalui statistik *haproxy* dengan alamat *domain syamok.unm.ac.id:1234/stats*. Adapun tampilan awal statistik *harproxy* ditunjukkan pada tabel 1 berikut.

Tabel 1. Tampilan awal halaman statistik *harproxy*

footer

	Queue			Session rate			Sessions						Bytes		Denied	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp
server1	0	0	-	0	0		0	0	-	0	0	?	0	0		0
server2	0	0	-	0	0		0	0	-	0	0	?	0	0		0
Backend	0	0		0	0		0	0	200	0	0	?	0	0	0	0

Denied		Errors			Warnings		Server								
Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
	0		0	0	0	0	1m45s UP	L4OK in 0ms	8	Y	-	1	1	0s	-
	0		0	0	0	0	1m49s UP	L4OK in 0ms	7	Y	-	0	0	0s	-
0	0		0	0	0	0	1m49s UP		15	2	0		0	0s	

Pengujian pertama dilakukan dengan mengakses secara berulang pada situs LMS moodle. Bobot yang diberikan adalah 8:7, sehingga didapat perubahan seperti pada tabel 2.

Tabel 2. Data setelah melakukan *refresh*

footer

	Queue			Session rate			Sessions						Bytes	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out
server1	0	0	-	0	54		0	9	-	67	67	1m32s	12 720	73 041
server2	0	0	-	0	46		0	6	-	58	58	1m32s	11 446	60 643
Backend	0	0		0	100		0	10	200	125	125	1m32s	24 166	133 684

Denied		Errors			Warnings		Server							
Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
	0		0	0	0	0	4m27s UP	L4OK in 0ms	8	Y	-	0	0	0s	-
	0		0	0	0	0	4m27s UP	L4OK in 0ms	7	Y	-	0	0	0s	-
0	0		0	0	0	0	4m27s UP		15	2	0		0	0s	

Tabel 3. Data setelah melakukan *login*

footer

	Queue			Session rate			Sessions						Bytes	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out
server1	0	0	-	0	54		0	7	-	333	333	3s	248 218	3 140 586
server2	0	0	-	0	46		0	6	-	96	96	3s	34 156	148 844
Backend	0	0		0	100		0	10	200	431	429	3s	283 268	3 289 856

Denied		Errors			Warnings		Server							
Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme
	0		0	0	0	0	23m25s UP	L4OK in 0ms	8	Y	-	12	4	0s
	0		0	0	0	0	20m34s UP	L4OK in 0ms	7	Y	-	6	2	0s
0	0		2	0	0	0	23m25s UP		15	2	0		3	8m4s

Tabel 2 menunjukkan hasil pengujian melakukan *refresh* saat kedua *server* menyala, diakses melalui *browser*, dan dibagi menggunakan algoritma *weighted round robin* dengan bobot 8:7. *Server* 1 memproses total *session* sebanyak 67 sesi dengan 12.720 *bytes* data masuk dan 73.041 *bytes* keluar, sedangkan *server* 2 memproses total *session* sebanyak 58 sesi dengan 11.446 *bytes* data masuk dan 60.643 *bytes* keluar.

Tabel 3 menunjukkan hasil pengujian setelah proses *login* pada *web* LMS dengan kedua *server* aktif. *Server* 1 memproses total *session* sebanyak 333 sesi dengan 248.218 *bytes* data masuk dan 3.140.586 *bytes* keluar, sedangkan *server* 2 memproses total *session* sebanyak 96 sesi dengan 34.156 *bytes* data masuk dan 148.844 *bytes* keluar. Selanjutnya dilakukan pengujian kedua dengan mengakses salah satu fitur dari LMS dengan membuka kalender dari *web* LMS dan data hasil dari pengujian tersebut seperti pada Tabel 4.

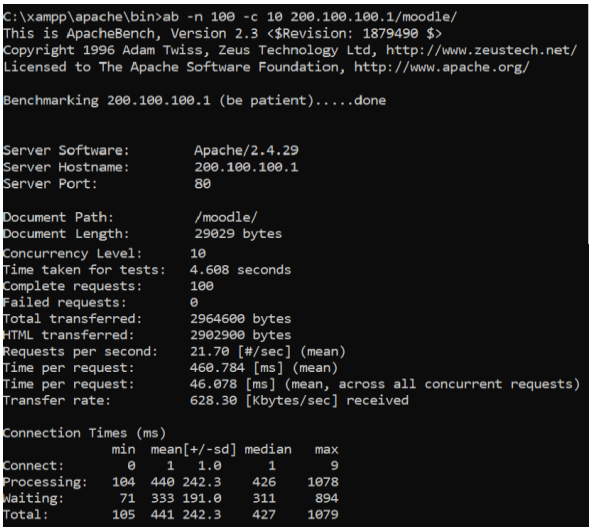
Tabel 4. Data pengujian kedua

footer

	Queue			Session rate			Sessions						Bytes	
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out
server1	0	0	-	0	41		0	6	-	777	777	2m45s	591 339	5 626 488
server2	0	0	-	0	40		0	6	-	562	559	58m24s	330 257	5 940 117
Backend	0	0		0	41		0	6	200	1 336	1 336	2m45s	921 596	11 566 605

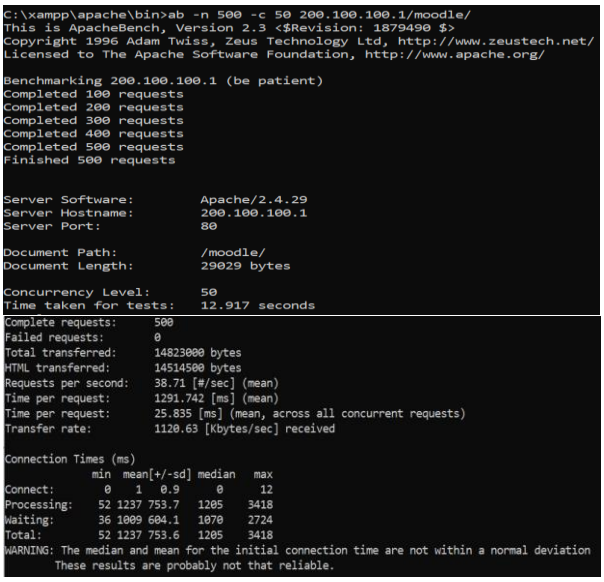
Denied		Errors			Warnings		Server								
Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
	0		0	0	0	0	1m25s UP	L4OK in 0ms	8	Y	-	20	7	0s	-
	0		1	0	3	0	49s UP	L4OK in 0ms	7	Y	-	17	5	0s	-
0	0		1	0	3	0	1m25s UP		15	2	0		8	32m57s	

Tabel 4 menunjukkan hasil pengujian kedua saat *user* mengakses kalender pada *web* LMS dengan kedua *server* aktif. *Server* 1 memproses total *session* sebanyak 777 sesi dengan 591.339 *bytes* data masuk dan 5.626.488 *bytes* keluar, sementara *server* 2 memproses total *session* sebanyak 562 sesi dengan 330.257 *bytes* masuk dan 5.940.117 *bytes* keluar. Pengujian ketiga dilakukan dengan menggunakan *apache bench* untuk mengirimkan sejumlah 100, 500 dan 1000 *request* ke *web* LMS dalam jumlah besar seperti pada gambar 12, 13, dan 14.



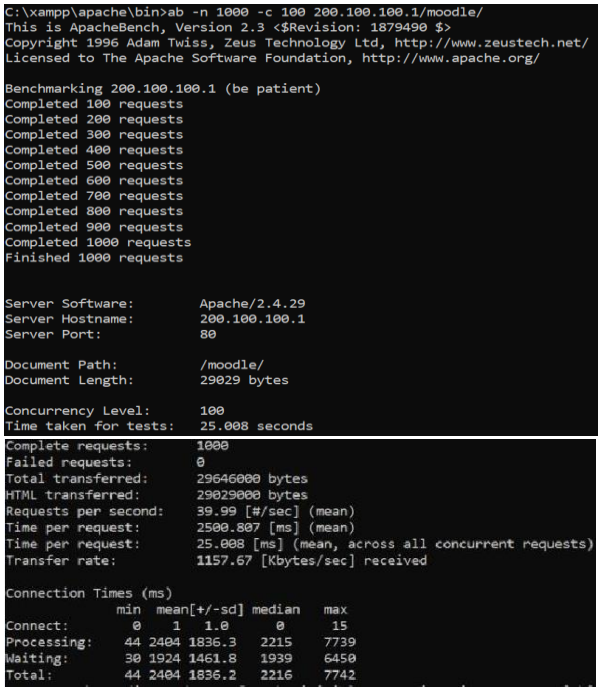
Gambar 12. Pengujian dengan 100 *request*

Pada Gambar 12 dilakukan pengujian pengiriman sebanyak 100 *request* dengan maksimal *request* yang berjalan bersamaan sebanyak 10 *request* ke *web* LMS. Pada Gambar 12 menampilkan kecepatan *request* yaitu sebesar 21.70 request/sec, waktu tercepat menanggapi *request* yaitu 105 *ms*.



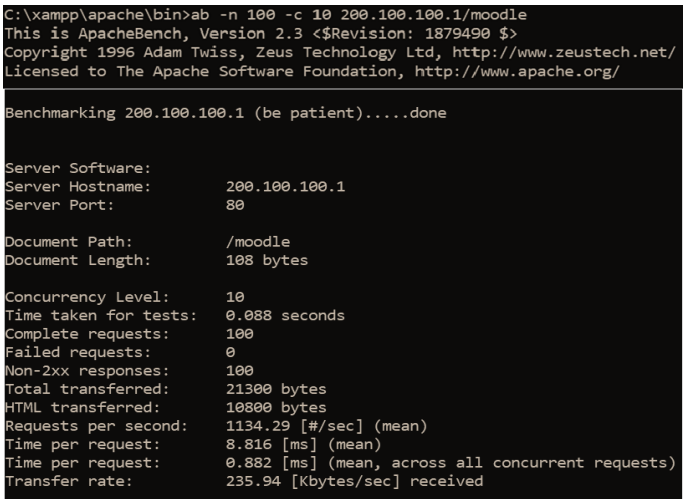
Gambar 13. Pengujian dengan 500 *request*

Pada Gambar 13 didapat data pengujian dalam kondisi kedua *server* menyala dan *user* mengirimkan *request* sebanyak 500 *request* dengan maksimal *request* yang berjalan bersamaan sebanyak 50 *request* ke *web* LMS. Pada Gambar 13 menampilkan kecepatan *request* yaitu sebesar 38.71 *request/sec*, waktu tercepat menanggapi *request* yaitu 52 *ms*.



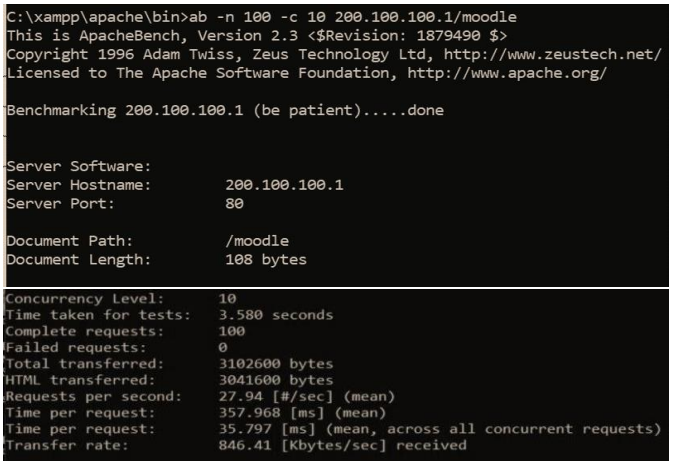
Gambar 14. Mengirimkan 1000 *request*

Pada Gambar 14 didapat data pengujian ketiga dalam kondisi kedua *server* menyala dan *user* mengirimkan *request* sebanyak 1000 *request* dengan maksimal *request* yang berjalan bersamaan sebanyak 100 *request* ke *web* LMS. Pada Gambar 14 menampilkan kecepatan *request* yaitu sebesar 39.99 *request/sec*, waktu tercepat menanggapi *request* yaitu 15 *ms*. Selanjutnya melakukan uji coba keempat dengan mengirimkan 100 *request* dengan *load balancing* dan tanpa *load balancing*. Percobaan keempat dapat dilihat pada Gambar 15 dan 16.



Gambar 15. Dengan *load balancing*

Pada gambar 15 didapat data pengujian dalam kondisi *server* menyala, di mana *user* mengirimkan 100 *request* dengan maksimal 10 *request* berjalan bersamaan ke *web* LMS. Gambar tersebut menampilkan kecepatan *request* sebesar 1134,29 *request/sec*, dengan rata-rata waktu per *request* di semua *concurrent request* sebesar 0,882 *ms*.



Gambar 16. Tanpa *load balancing*

Pada gambar 16 didapat data pengujian dalam kondisi *server* menyala, di mana *user* mengirimkan 1000 *request* dengan maksimal 10 *request* berjalan bersamaan ke *web* LMS. Gambar tersebut menampilkan kecepatan *request* sebesar 27,94 *request/sec*, dengan rata-rata waktu per *request* di semua *concurrent request* sebesar 357.97 *ms*.

Pembahasan

Pada penelitian ini, terdapat beberapa tahap yang dilalui sebelum dilakukan pengujian. Pada tahap pertama terdapat tahap analisis kebutuhan, aplikasi dan perangkat keras yang mendukung pembuatan prototipe sistem *load balancing* dipilih, di mana *haproxy* versi 2.3 dipilih karena bersifat gratis dan *open source* serta dapat berjalan pada sistem operasi *Linux*. *haproxy* juga dapat meminimalkan *downtime* dengan mendistribusikan beban secara efisien antara *server*, serta menyediakan fitur pemantauan statistik untuk memonitor akses *user*. Selain itu, LMS *Moodle* versi 3.8 dipilih sebagai aplikasi *web server* karena kemudahan penggunaannya dan sifatnya yang *open source*. *MariaDB* versi 10.1 dipilih sebagai sistem manajemen database karena kompatibel dengan *Moodle* dan dapat berjalan pada *Linux*. Kapasitas *server* dihitung berdasarkan aturan umum bahwa 1 GB RAM dapat menangani sekitar 50 *concurrent users*, sehingga dengan RAM 512 MB pada setiap *server* virtual, sistem dapat menangani sekitar 50 *user* bersamaan.

Kemudian tahap berikutnya ada tahap perancangan sistem, dengan memilih algoritma load balancing WRR (*Weighted Round Robin*), yang mempertimbangkan kapasitas masing-masing *server*. Dengan WRR, beban akan dibagi sesuai bobot yang ditetapkan, sehingga *server* dengan kapasitas lebih besar menangani lebih banyak permintaan dibandingkan server yang lebih kecil. Pemilihan algoritma ini bertujuan untuk meningkatkan efisiensi dalam distribusi beban antar *server* dan memaksimalkan penggunaan kapasitas *server*. Selain itu, konfigurasi *server* juga sangat penting. Pembagian bobot antara *server* pertama dan kedua dilakukan, dengan bobot 8 untuk *server* pertama dan 7 untuk *server* kedua, berdasarkan kapasitas masing-masing. Ini memastikan bahwa *request* dibagi secara proporsional sesuai dengan kemampuan *server*. Konfigurasi jaringan antara *server web*, *database*, dan *load balancer* juga dilakukan, dengan menggunakan alamat IP yang sesuai dan memastikan *server* dapat saling terhubung. *haproxy* digunakan sebagai *load balancer* untuk menerapkan algoritma WRR, yang secara otomatis mendistribusikan *request* ke *server* yang sesuai.

Selain konfigurasi *load balancing*, pengaturan *database* juga penting agar aplikasi *Moodle* pada LMS dapat berjalan dengan baik. *Database MariaDB* dikonfigurasi dengan *engine InnoDB* dan format file *Barracuda* untuk memastikan performa yang optimal. Pembuatan *database Moodle* dan *user* dengan hak akses penuh juga dilakukan, yang memungkinkan *server web* untuk terhubung dan mengakses data yang diperlukan. Pada sisi *server web*, aplikasi *Moodle* diinstal dan dikonfigurasi di *Apache Web Server* dengan mengatur *Virtual Host* untuk mengarahkan *traffic* ke direktori yang tepat.

Adapun pada tahap pengujian, dilakukan empat tahap pengujian. Pengujian pertama dilakukan dengan mengakses situs LMS *Moodle* secara berulang. Tujuan pengujian ini adalah untuk melihat apakah data sesi yang diterima oleh *server* terbagi sesuai dengan bobot yang diberikan pada dua unit *server*, yaitu 8:7. Dalam pengujian ini, kedua *server* dalam kondisi aktif, dan akses dilakukan melalui *browser* untuk me-*refresh* halaman LMS. Akses ini akan dibagi secara berulang kepada dua unit *server* sesuai dengan algoritma *Weighted Round Robin*. Pembagian sesi dimulai dengan *Server* 1, diikuti oleh *Server* 2, sesuai dengan bobot yang ditetapkan, yaitu 8:7. Pada *Server* 1, diproses total 67 sesi dengan data masuk sebanyak 12.720 *bytes* dan data keluar sebanyak 73.041 *bytes*. Sementara itu, pada *Server* 2, diproses total 58 sesi dengan data masuk sebanyak 11.446 *bytes* dan data keluar sebanyak 60.643 *bytes*.

Selanjutnya didapat data pengujian setelah *user* melakukan login ke *web* LMS. Pada tahap ini, kedua *server* masih dalam kondisi aktif. Pada *Server* 1, diproses total 333 sesi dengan data masuk sebanyak 248.218 *bytes* dan data keluar sebanyak 3.140.586 *bytes*. Sementara itu, pada *Server* 2, diproses total 96 sesi dengan data masuk sebanyak 34.156 *bytes* dan data keluar sebanyak 148.844 *bytes*. Penambahan total sesi dan *bytes* yang lebih besar pada pengujian ini disebabkan oleh proses *login* yang menghasilkan lebih banyak permintaan dibandingkan dengan proses *refresh* halaman sebelumnya. Perbandingan antara total sesi, total *bytes* masuk, dan total *bytes* keluar dapat dilihat pada Tabel 5.

Tabel 5. Pembanding setelah melakukan *refresh* dan *login*

Pembanding	***Refresh***	***Login***
Session total	125 sesi	431 sesi
Bytes in total	24.166 *bytes*	283.268 *bytes*
Bytes Out total	133.684 *bytes*	3.289.856 *bytes*

Selanjutnya pengujian kedua dilakukan dengan mengakses salah satu fitur dari LMS dengan membuka kalender dari *web* LMS. didapat data pengujian kedua dalam kondisi kedua *server* menyala dan *user* mengakses kalender dari *web* LMS. Pada *server* satu memproses total *session* sebanyak 777 sesi dengan total *bytes* data 591.339 *bytes* masuk dan 5.626.488 *bytes* keluar. Pada *server* dua memproses total *session* sebanyak 562 sesi dengan total *bytes* data 330.257 *bytes* masuk dan 5.940.117 *bytes* keluar. Secara keseluruhan, hasil pengujian ini menunjukkan bahwa kedua *server* berfungsi dengan baik dalam menangani beban yang dibagi, meskipun ada sedikit perbedaan dalam hal jumlah sesi yang diproses dan data yang diproses. Ini menunjukkan bahwa sistem *load balancing* yang diterapkan dapat mendistribusikan beban secara efektif sesuai dengan kapasitas masing-masing *server*.

Kemudian pengujian ketiga dilakukan dengan menggunakan *apache bench* untuk mengirimkan sejumlah 100, 500 dan 1000 *request* ke *web* LMS dalam jumlah besar. Adapun perbandingan antara ketiga 100, 500 dan 1000 *request* dapat dilihat pada tabel 6.

Tabel 6. Perbandingan 100, 500 dan 1000 *request*

Pembanding	100 *request*	500 *request*	1000 *request*
Session total	105 sesi	605 sesi	1.500 sesi
Bytes in total	11. 616 *bytes*	55.616 *bytes*	132.000 *bytes*
Bytes out total	2.974.266 *bytes*	14.823.000 *bytes*	44.469.000 *bytes*

Pada tabel 6 didapat hasil pengujian dengan pengiriman sebanyak 100 *request* dengan maksimal *request* yang berjalan bersamaan sebanyak 10 *request* ke *web* LMS. Didapat kecepatan *request* yaitu sebesar 21.70 *request/sec*, serta waktu tercepat menanggapi *request* yaitu 105 *ms*. Kemudian pada kondisi kedua *server* menyala dan *user* mengirimkan *request* sebanyak 500 *request* dengan maksimal *request* yang berjalan bersamaan sebanyak 50 *request* ke *web* LMS. Didapatkan kecepatan *request* yaitu sebesar 38.71 *request/sec*, serta waktu tercepat menanggapi *request* yaitu 52 *ms*.

Selanjutnya dalam kondisi kedua *server* menyala dan *user* mengirimkan *request* sebanyak 1000 *request* dengan maksimal *request* yang berjalan bersamaan sebanyak 100 *request* ke *web* LMS. Didapatkan kecepatan *request* yaitu sebesar 39.99 *request/sec*, serta waktu tercepat menanggapi *request* yaitu 15 *ms*. Pada tabel 6, hasil analisis pengujian memberitahukan bahwa apabila *user* melakukan *request* yang banyak maka *server* yang diberikan *load balancing* akan mengakses data tersebut dengan cepat, hasil tersebut dapat dilihat dengan angka 1.500 sesi dari *session* total, 132.000 dari *bytes* in total dan 44.469.000 dari *bytes out* total, data tersebut dihasilkan saat melakukan pengujian pada *apache bench* dengan 1000 *request*. Pada tabel juga dapat dilihat bahwa semakin banyak *request* yang dilakukan oleh *user* maka data yang dikeluarkan banyak dan *request* yang dilakukan cepat.

Selanjutnya terdapat pengujian keempat dengan mengirimkan 100 *request* dengan *load balancing* dan tanpa *load balancing*. Pada pengujian ini diperoleh hasil dari pengujian dengan *load balancing* dalam kondisi *server* menyala, di mana *user* mengirimkan 100 *request* dengan maksimal 10 *request* berjalan bersamaan ke *web* LMS. Didapatkan kecepatan *request* sebesar 1134,29 *request/sec*, dengan rata-rata waktu per *request* di semua *concurrent request* sebesar 0,882 *ms*. Sedangkan pada pengujian tanpa *load balancing*, diperoleh hasil pengujian dalam kondisi *server* menyala, di mana *user* mengirimkan 1000 *request* dengan maksimal 10 *request* berjalan bersamaan ke *web* LMS. Didapatkan kecepatan *request* sebesar 27,94 *request/sec*, dengan rata-rata waktu per *request* di semua *concurrent request* sebesar 357.97 *ms*.

Pada hasil pengujian yang diperoleh dapat dilihat perbandingan *apache bench* pengiriman 100 *request* dengan 10 *request* yang berjalan bersamaan antara *prototype* sistem tanpa menggunakan *load balancing* dengan *prototype* sistem menggunakan *load balancing* seperti yang ditampilkan pada tabel 7.

Tabel 7. Perbandingan hasil *apache bench* 100 *request* dengan dan tanpa *load balancing*

Pembanding	Dengan *Load Balancing*	Tanpa *Load Balancing*

Kecepatan pengembalian *request* (*request/sec*)	1134.3 *request/sec*	27.94 *request/sec*
Rata-rata waktu per *request* di semua *concurrent request* (ms)	8.816 ms	357.97 ms

Pada tabel 7, diperoleh perbandingan pada kecepatan pengembalian *request* LMS yang menggunakan *load balancing* lebih cepat yaitu sebesar 1134.3 *request/sec*. Kemudian untuk rata-rata waktu menanggapi *request* di semua *concurrent request* LMS dengan *load balancing* juga lebih cepat sekitar 8.816 *ms* dibandingkan dengan tanpa menggunakan *load balancing*.

Kesimpulan

Berdasarkan hasil pembuatan prototype *load balancing* menggunakan algoritma WRR, dapat disimpulkan bahwa: Hasil *login* dan *refresh* berfungsi dengan baik, terbukti dengan halaman *web* yang tampil dengan baik dan data yang diperoleh dari statistik *haproxy*, yaitu 24.166 *bytes* untuk *refresh* dan 283.268 *bytes* untuk *login*. Perbandingan 100, 500, dan 1000 *request* menghasilkan 105 sesi untuk 100 *request*, 605 sesi untuk 500 *request*, dan 1500 sesi untuk 1000 *request*, dengan waktu 44 *ms* untuk 100 *request*, 52 *ms* untuk 500 *request*, dan 105 *ms* untuk 1000 *request*. Hal ini menunjukkan bahwa semakin banyak *request*, semakin lama waktu pemrosesan. Data dari statistik *haproxy* menunjukkan total 1.336 sesi dalam 2 menit 45 detik, dengan halaman kalender yang ditampilkan dengan baik. Analisis *load balancing* menunjukkan waktu *respons* 882 *ms* dengan *load balancing* dan 358 *ms* tanpa *load balancing*, yang berarti penggunaan *load balancing* mempercepat akses data dibandingkan tanpa *load balancing*.

Daftar Pustaka

Agung, M. (2022). Pengembangan Aplikasi Lunak Untuk Monitoring Kuliah Daring Dalam Upaya Penanggulangan Wabah Covid-19. *Pengembangan Aplikasi Lunak Untuk Monitoring Kuliah Daring Dalam Upaya Penanggulangan Wabah Covid-19*, *1*(2), 125–134.

Azhar, R., Santoso, H., & Krismono, B. (2022). Pengaruh Implementasi Kernel Based Virtual Machine Pada Server Vps Terhadap Pemakaian Cpu Memory Dan Harddisk. *Jurnal Informatika Dan Rekayasa Elektronik*, *5*(1), 140–152. https://doi.org/10.36595/jire.v5i1.522

Baharuddin, N., Muhammad Yahya, & Muh. Yusuf Mappease. (2023). Efektivitas Penggunaan SYAM-OK Sebagai LMS Dalam Mendukung Proses Pembelajaran Daring di Prodi Pendidikan Teknik Informatika Dan Komputer Universitas Negeri Makassar. *TEKNOVOKASI : Jurnal Pengabdian Masyarakat*, *1*(1), 13–24. https://doi.org/10.59562/teknovokasi.v1i1.10

Hakim, D. K., Yulianto, D. Y., & Fauzan, A. (2019). Pengujian Algoritma Load Balancing pada Web Server Menggunakan NGINX. *JRST (Jurnal Riset Sains Dan Teknologi)*, *3*(2), 85.

https://doi.org/10.30595/jrst.v3i2.5165

Hanafiah, A. (2021). Implementasi Load Balancing Dengan Algoritma Penjadwalan Weighted Round Robin Dalam Mengatasi Beban Webserver. *IT Journal Research and Development*, *5*(2), 226–233. https://doi.org/10.25299/itjrd.2021.vol5(2).5795

Hidayat, A. S., Widodo, A. E., Kencono, A., & Nuryamin, Y. (2021). Implementasi Load Balancing dengan metode PCC pada Balai Besar Pelatihan Kesehatan (BBPK) Jakarta. *Jurnal Sains Dan Manajemen*, *9*(1), 101–112.

Hidayat, M. . (2022). *Penerapan Load Balancing Pada Server Menggunakan Algoritme Least Connection di Sekolah Vokasi IPB.*

Samad, P. I. S. (2021). the Impact of E-Learning With (Syam-Ok) During Pandemic on English Learning Results of Students At Electronic Engineering Education Department of Makassar State University. *Jurnal Nalar Pendidikan*, *9*(2), 87. https://doi.org/10.26858/jnp.v9i2.24170