BAB III
METODE PENELITIAN

Penelitian ini akan menerapkan *load balancing* pada LMS berbasis *cloud computing*. Penelitian ini menerapkan metode kuantitatif dengan dengan eksperimen dan analisis data dari hasil pengujian.

3.1 PERANGKAT YANG DIGUNAKAN

3.1.1 PERANGKAT KERAS (*HARDWARE*)

Penelitian ini memakai perangkat keras sebagai *hypervisor* dalam menjalankan virtualisasi pada simulasi *load balancing* LMS, spesifikasi perangkat keras ada dalam tabel 3.1 sebagai berikut.

Tabel 3.1 Spesifikasi *hardware*.

Sistem Operasi	Windows 10
Processor	Intel® Core™ i7-9700 CPU 3.60 GHz
RAM	32 GB

3.1.2 PERANGKAT LUNAK (*SOFTWARE*)

3.1.2.1 PERANGKAT VIRTUALISASI

Sebagai perangkat lunak dalam menjalankan simulasi *load balancing* pada LMS, beberapa perangkat lunak yang digunakan yaitu satu sebagai *client*, satu *Openstack server*, tiga *Server* LMS pada *instance Openstack*, dan satu sebagai *load balancer*. Pada tabel 3.2 dicantumkan spesifikasi perangkat lunak yang diterapkan pada penelitian ini.

Tabel 3.2 Spesifikasi perangkat virtualisasi.

CLIENT	Sistem Operasi	Ubuntu 18.04 LTS
	RAM	1 GB
	HDD	10 GB
	IP	172.24.4.48
OPENSTACK SERVER	Sistem Operasi	Ubuntu 18.04 LTS
	RAM	20 GB
	HDD	200 GB
	IP	10.212.16.68

Tabel 3.2 Spesifikasi perangkat virtualisasi.

LMS	Sistem Operasi	Ubuntu 18.04 LTS
	RAM	2 GB
	HDD	20 GB
	IP	172.24.4.128
LMS	Sistem Operasi	Ubuntu 18.04 LTS
	RAM	2 GB
	HDD	20 GB
	IP	172.24.4.7
LMS	Sistem Operasi	Ubuntu 18.04 LTS
	RAM	2 GB
	HDD	20 GB
	IP	172.24.4.139
LOAD BALANCER	Sistem Operasi	Ubuntu 18.04 LTS
	RAM	2 GB
	HDD	10 GB
	IP	172.24.4.182

3.1.2.2 *SOFTWARE TOOLS*

Penelitian ini menggunakan *software tool* untuk melakukan implementasi *load balancing* pada LMS. *Tools* yang dipakai dalam penerapan *load balancing* pada LMS dilampirkan dalam tabel 3.3 sebagai berikut.

Tabel 3.3 Perangkat lunak.

NO	SOFTWARE	FUNGSI
1	*VirtualBox*	Virtualisasi
2	*Openstack*	*Cloud Computing*
3	*HAProxy*	*Load Balancer*
4	*Chamilo*	LMS

3.1.2.3 *SOFTWARE* PENGUJIAN PERFORMA LMS

1) HTTPERF

HTTPerf adalah untuk menguji performa *web server*. *HTTPerf* menyediakan tools pengujian yang fleksibel dalam protokol HTTP dalam mengukur kinerja *web server*. *HTTPerf* menyediakan alat pengujian yang berkinerja tinggi dan memfasilitasi *micro* and *macro* level *benchmarks* [28]. Sistem *web* terdiri dari *web server*, *client*, dan jaringan yang menghubungkan *client* ke *server*. Protokol yang digunakan antara *client* dan *server* untuk berkomunikasi adalah HTTP.

```
httperf --hog --server 172.24.4.182 --port 80 --num-conn 1000 --rate 100
```

Perintah diatas merupakan perintah pengujian menggunakan *HTTPerf*, pengujian berjalan pada port 80. Alamat IP *server* dalam pengujian sederhana ini menggunakan alamat IP load balancer . Jumlah koneksi yang digunakan adalah 100 koneksi per detik. Pengujian ini melibatkan inisiasi total 1000 paket koneksi TCP dan pada setiap koneksi, satu panggilan HTTP dilakukan (panggilan terdiri dari mengirim permintaan dan menerima jawaban) [29].

Total: connections 1000 requests 1000 replies 1000 test-duration 77.881 s
Connection rate: 12.8 conn/s (77.9 ms/conn, <=969 concurrent connections)
Connection time [ms]: min 3804.4 avg 42674.4 max 68346.2 median 50046.5 stddev 16511.5
Connection time [ms]: connect 9.0
Connection length [replies/conn]: 1.000
Request rate: 12.8 req/s (77.9 ms/req)
Request size [B]: 65.0
Reply rate [replies/s]: min 1.4 avg 13.1 max 49.0 stddev 13.3 (15 samples)
Reply time [ms]: response 42565.0 transfer 100.7
Reply size [B]: header 186.0 content 8208.0 footer 0.0 (total 8394.0)
Reply status: 1xx=0 2xx=233 3xx=0 4xx=0 5xx=767
CPU time [s]: user 5.01 system 72.25 (user 6.4% system 92.8% total 99.2%)
Net I/O: 106.1 KB/s (0.9*10^6 bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Ada enam kelompok statistik : hasil keseluruhan (“Total”), hasil terkait koneksi (“*Connection*”), hasil yang berkaitan dengan jumlah permintaan HTTP (“*Request*”), hasil yang berkaitan dengan balasan yang diterima dari *server* (“*Reply*”), hasil lain yang berkaitan dengan penggunaan CPU dan jaringan (“Net

I/O") yang tak kalah pentingnya yaitu ringkasan kesalahan yang ditemui ("Error"). Penjelasan tiap kelompok statisik akan dijelaskan sebagai berikut.

1) *Total section*

   Bagian ini merangkum berapa banyak koneksi TCP yang dimulai oleh *HTTPerf*, berapa banyak permintaan yang dikirimkannya, berapa banyak balasan yang diterimanya, dan berapa total durasi pengujian.

2) *Connection section*

   Bagian ini menyampaikan informasi terkait koneksi TCP yang dihasilkan oleh alat tersebut. Secara khusus, baris "*Connection rate*" menunjukkan bahwa koneksi baru dimulai dengan kecepatan 77.9 koneksi per detik. Angka terakhir pada baris ini menunjukkan bahwa paling banyak 969 sambungan dibuka pada waktu tertentu. Baris pertama "*Connection time*" memberikan statistik untuk koneksi yang berhasil. Masa pakai koneksi adalah waktu antara koneksi TCP dimulai dan waktu koneksi ditutup. Sambungan dianggap berhasil jika memiliki setidaknya satu panggilan yang berhasil diselesaikan. Statistik berikutnya di bagian ini adalah waktu rata-rata yang diperlukan untuk membuat koneksi TCP. Hanya jumlah koneksi TCP yang berhasil yang dihitung. Baris terakhir di bagian ini yaitu "*Connection length*" Ini memberikan jumlah rata-rata balasan yang diterima pada setiap koneksi yang menerima setidaknya satu balasan

3) *Request section*

   Baris "*Request rate*" memberikan tingkat di mana permintaan HTTP diselesaikan per detiknya. Baris "*Request size*" memberikan ukuran rata-rata permintaan HTTP dalam *Byte*.

4) *Reply section*

   Untuk pengukuran sederhana, bagian ini sering kali paling menarik karena baris "*Reply rate*" memberikan berbagai statistik untuk tingkat balasan. Baris "*Reply time*" memberikan informasi tentang berapa lama waktu yang dibutuhkan *server* untuk merespons dan berapa lama waktu yang dibutuhkan untuk menerima balasan. Baris berikutnya, "*Request size*" berisi statistik tentang ukuran rata-rata balasan semua angka dalam *byte* yang dilaporkan. Secara khusus, baris tersebut mencantumkan panjang rata-rata *header* balasan,

konten dan *footer* (HTTP/1.1 menggunakan *footer* untuk mewujudkan pengkodean transfer. Baris terakhir di bagian ini adalah histogram dari kode status utama yang diterima dalam balasan dari *server*.

5) *Miscellaneous section*

   Bagian ini dimulai dengan ringkasan penggunaan CPU pada mesin klien. Baris berlabel "Net I/O" memberikan *throughput* jaringan rata-rata dalam *kilobyte* per detik (di mana satu *kilobyte* adalah 1024 *byte*) dan dalam megabit per detik (di mana satu megabit adalah 10^6 *bit*).

6) *Errors section*

   Bagian terakhir berisi statistik tentang kesalahan yang ditemukan selama pengujian. Dalam contoh, dua baris berlabel "Error" menunjukkan bahwa ada total nihil kesalahan apabila hasil yang didapatkan maka kesalahan tersebut disebabkan oleh *server* yang menolak untuk menerima koneksi (*connrefused*). Pada bagian *client-timo* adalah berapa kali koneksi yang gagal, socket-timo adalah berapa kali koneksi TCP yang gagal dengan *socket-level timeout* dan *connrefused* adalah berapa kali koneksi TCP yang gagal karena koneksi ditolak oleh *server*.

3.2 ALUR PENELITIAN

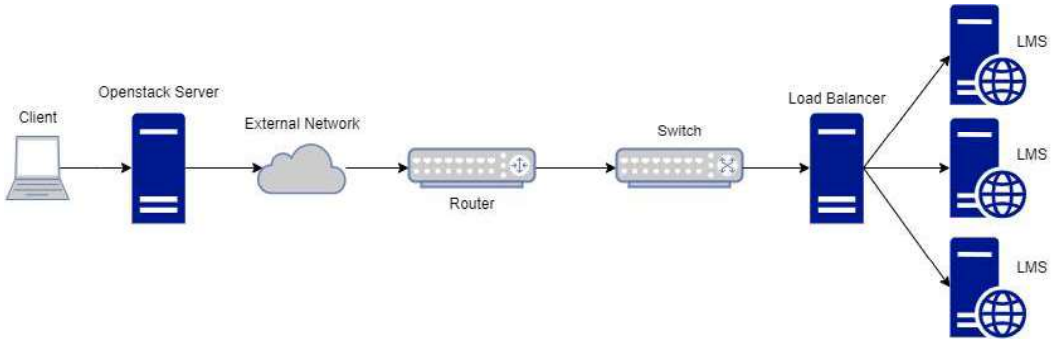
Alur penelitian yang dilakukan pada penelitian ini diawali dengan melakukan studi literatur dengan membaca beberapa penelitian terdahulu. Selanjutnya melakukan perancangan topologi jaringan sebagai rancangan awal sebelum melakukan implementasi *load balancing* pada LMS. Selanjutnya melakukan implementasi *load balancing* pada LMS, dimulai dengan membangun infrastruktur *cloud computing*, menerapkan LMS dan *load balancer* pada *instance Openstack*. Selanjutnya melakukan perancangan skenario pengujian, lalu melakukan analisis hasil data berdasarkan parameter luaran *throughput, packet loss, response time*, dan CPU *usage*.

Gambar 3.1 Alur penelitian.

Alur penelitian yang dilakukan digambarkan pada Gambar 3.1. Hal pertama yang dilakukan membuat topologi jaringan sebelum membuat simulasi, yang bertujuan sebagai gambaran dalam melakukan implementasi *load balancing* pada LMS. Selanjutnya menerapkan infrastruktur *cloud computing* dengan *devstack* yang berjalan pada OS *Ubuntu Server* 20.04. Setelah infrastruktur *cloud* selesai dikerjakan, hal selanjutnya yang perlu dilakukan yaitu meluncurkan *instance*

sebanyak empat instance dengan tiga sebagai LMS dan satu sebagai *load balancer*. Selanjutnya konfigurasi *load balancing*, apabila berhasil maka selanjutnya adalah melakukan pengujian berdasarkan parameter luaran yaitu *throughput, packet loss, response time* dan CPU *usage*. Setelah data performansi dikumpulkan, maka selanjutnya menganalisisis hasil pengujian yang dilakukan untuk mendapatkan hasil performansi pada *load balancing* pada LMS.

3.3 TOPOLOGI JARINGAN



Gambar 3.2 Topologi jaringan.

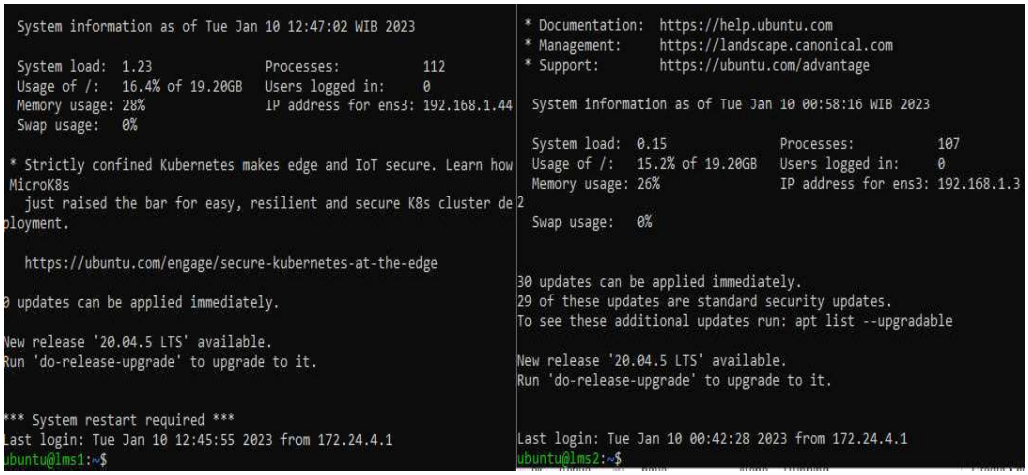
Sebelum melakukan penelitian ini, diperlukan topologi jaringan sebagai gambaran implementasi load balancing pada LMS digambarkan pada Gambar 3.2. Pada topologi ini menggunakan satu *client* sebagai tempat untuk melakukan pengujian performansi, satu *server* untuk menerapkan *cloud computing* dengan Openstack, satu *external network* yang berfungsi untuk sebagai koneksi jaringan pada *virtual machine* ke jaringan internet. Satu *router* yang terhubung dengan *switch* untuk menghubungkan semua jaringan pada *virtual machine*.

3.4 SKENARIO PENGUJIAN

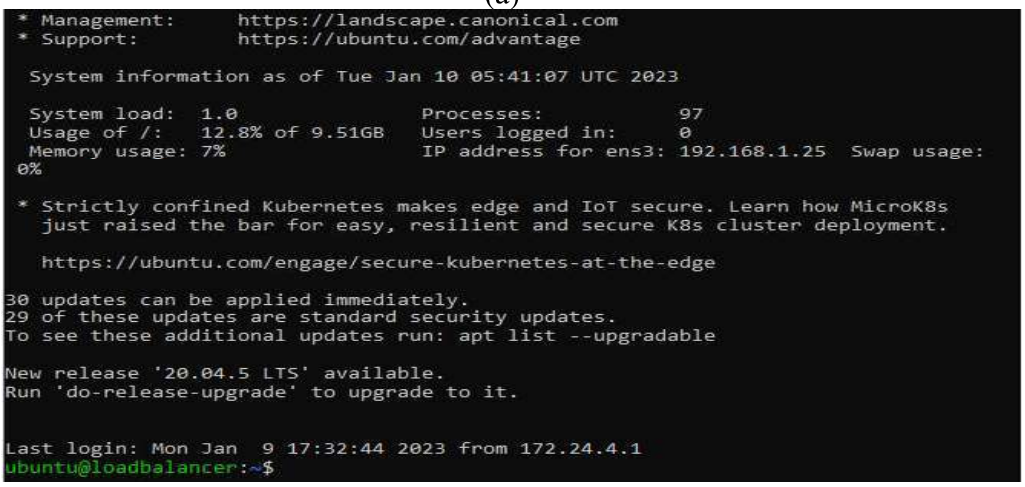
3.4.1 IMPLEMENTASI JARINGAN

Implementasi jaringan dalam penelitian ini menggunakan tiga *server* LMS sebagai layanan *e-learning*, satu *client* sebagai simulasi melakukan pengujian jaringan. *Load balancing* pada *server* membagikan beban kerja pada setiap *server* LMS. *Switch* menghubungkan seluruh jaringan pada *server* LMS, kemudian terhubung ke *router* yang menghubungkan jaringan yang berbeda. *External network* menghubungkan *instance* ke jaringan internet. *Client* digunakan untuk melakukan pengujian performansi LMS dengan parameter luaran yaitu *throughput*

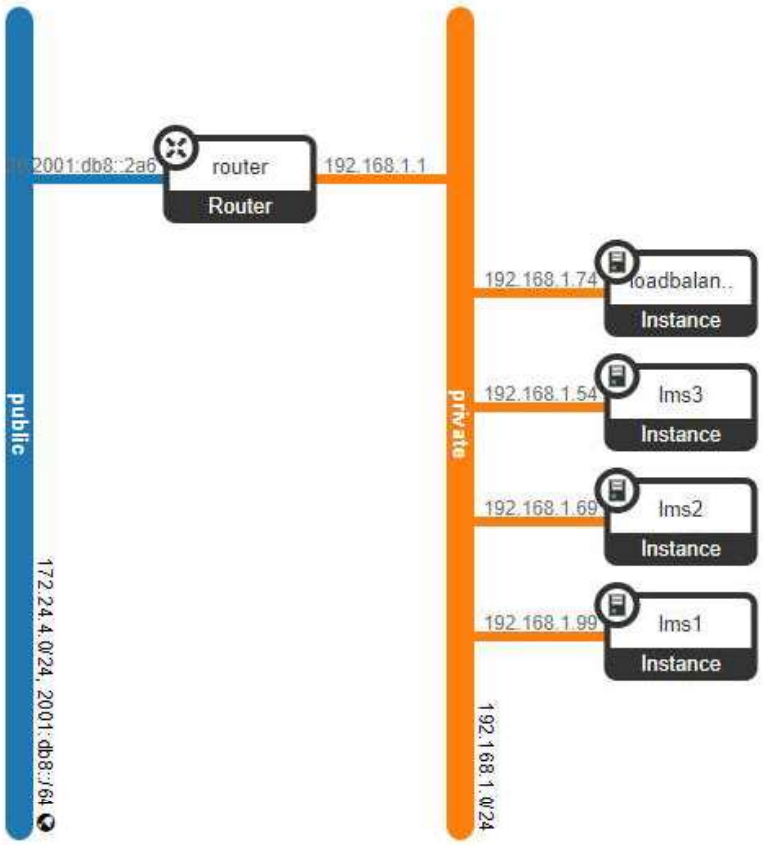
dan *packet loss, response time* dan CPU *usage* menggunakan *HTTPerf* untuk menguji *server* LMS.



(a)



(b)



(c)

Gambar 3.3 Hasil implementasi load balancing LMS pada *Openstack*.

(a) Implementasi LMS pada instance *Openstack*.

(b) Implementasi *load balancer*.

(c) Topologi jaringan dari *Openstack*.

3.4.2 IMPLEMENTASI *LOAD BALANCING*

Implementasi *load balancing* pada penelitian ini menggunakan *software HAPROXY* dengan membagikan beban pada dua *server* LMS. Sebelum mengimplementasikan *load balancing*, diperlukan konfigurasi *Openstack* dan konfigurasi *Chamilo* sebagai *software* LMS.

3.4.2.1 KONFIGURASI *CLOUD COMPUTING*

Infrastruktur *cloud computing* dalam penelitian ini menggunakan *Openstack devstack*, yang menjalankan *instance* sebagai *server* virtual dalam menerapkan *load balancing* pada LMS. Konfigurasi dilakukan pada *file* local.conf yang terletak pada direktori /home/devstack. Konfigurasi pada *file* local.conf

diperlukan untuk menentukan *username*, *password* dan *host* IP yang digunakan untuk mengakses *Openstack dashboard*. Setelah selesai melakukan konfigurasi maka selanjutnya adalah memulai proses instalasi *Openstack* dengan perintah /.stack.sh.

```
# Password for KeyStone, Database, RabbitMQ and Service
ADMIN_PASSWORD=admin
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
# Host IP - get your Server/VM IP address from ip addr command
HOST_IP=10.212.16.68
```

Gambar 3.4 Konfigurasi *Openstack* pada file local.conf.

3.4.2.2 KONFIGURASI *CHAMILO* LMS

Konfigurasi *Chamilo* dilakukan pada tiga *instance Openstack* yang telah dipersiapkan sebagai *server* LMS. Konfigurasi awal yaitu melakukan instalasi beberapa *software* seperti *Apache* sebagai *web server*, *software* PHP untuk menyimpan konten kursus dan data lainnya serta MySQL sebagai *database*. Setelah semua *software* diinstalasi, selanjutnya melakukan instalasi *Chamilo* yang digunakan sebagai *software* LMS. Konfigurasi yang diperlukan dalam instalasi *Chamilo* ditampilkan pada Gambar 3.5. Setelah melakukan instalasi *Chamilo*, langkah selanjutnya yaitu melakukan *setting Chamilo* melalui *browser*.

```
$ cd /tmp && wget
https://github.com/chamilo/chamilo-
lms/releases/download/v1.11.6/chamilo
-1.11.6-php7.zip
$ unzip chamilo-1.11.6-php7.zip
```

Gambar 3.5 Perintah instalasi *Chamilo* LMS.

3.4.2.3 KONFIGURASI DAN UJI AKSES *LOAD BALANCING*

Load balancing dikonfigurasi pada satu *instance Openstack* yang dipersiapkan sebagai *load balancer*. *Load balancing* memakai *HAPROXY* yang telah diinstalasi pada *instance load balancer*. Konfigurasi yang dilakukan pada *HAPROXY* adalah dengan menambahkan perintah pada direktori

/etc/haproxy/haproxy.cfg yang bertujuan untuk menentukan target *server* yang akan diseimbangkan bebannya dan menentukan algoritma pada *load balancing*.

```
frontend header
bind *:80
mode http
default_backend footer
backend footer
mode http
balance leastconn
server lms1 172.24.4.128:80 check
server lms2 172.24.4.7:80 check
server lms3 172.24.4.139:80 check
```

Gambar 3.6 Konfigurasi *load balancing* dengan *HAPROXY*.

Sistem *load balancing* perlu diuji akses untuk melihat apakah sistem berhasil. Uji akses dilakukan dengan memakai alamat *floating* IP *load balancer* pada *browser*. Ketika IP *load balancer* di akses pada *browser* maka dapat melihat LMS mana yang sedang diakses. Seperti pada Gambar 3.7, ketika mengakses IP *load balancer* pada *browser*, LMS yang ditampilkan adalah LMS 3.

Gambar 3.7 Tampilan akses LMS.

Selanjutnya melakukan uji akses *user* pada *browser*, bertujuan untuk menguji apakah sistem *load balancing* pada LMS sudah bekerja sesuai dengan algoritma *least connection*. *User* mengakses dengan lima PC yang berbeda-beda, yang masing-masing nya akan mengakses alamat IP *load balancer*. Uji coba sistem *load balancing* dilakukan dengan menggunakan lima PC sebagai *user* yang akan mengakses alamat IP *floating* pada *browser*, hasil uji akses pada *user* akan ditampilkan pada Tabel 3.4.

Tabel 3.4 Hasil uji *user* ke alamat IP *load balancer*.

User	LMS 1	LMS 2	LMS 3
1			
2			
3			
4			
5			

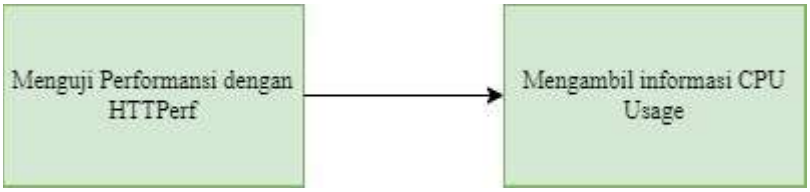
3.4.3 PENGUJIAN PERFORMANSI LMS

Pengujian performansi dengan parameter luaran *throughput*, *response time*, *packet loss* dan CPU *usage* bertujuan untuk melihat kualitas jaringan pada LMS saat permintaan dikirimkan ke *server* LMS. Hasil pengujian parameter QoS dilakukan dengan melalui tiga variabel pengujian dengan mengirimkan 1000, 1500, 2500, 3000 permintaan, *request* per detik setiap koneksi sebesar 100 permintaan per detik. Pengujian akan dilakukan sebanyak 30 kali untuk mendapatkan hasil pengujian yang valid dan tepat. Jumlah koneksi, *request* per detik dan jumlah pengujian yang akan dilakukan dilampirkan pada Tabel 3.5.

Tabel 3.5 Jumlah permintaan, *request* per detik dan jumlah pengujian.

No	Jumlah permintaan	*Request* per detik
1	1000	100
2	1500	100
3	2000	100
4	2500	100
5	3000	100

Pengujian LMS yang dilakukan dengan mengirimkan *request* menggunakan *HTTPerf*. Pengujian dilakukan dengan perintah "num-conns" untuk menentukan jumlah koneksi yang dikirimkan serta "*rate*" untuk menentukan permintaan per detik. Penggunaan *HTTPerf* untuk mendapatkan hasil pengujian *throughput*, *response time*, *packet loss* serta menggunakan perintah "sar 2" di setiap *server* untuk memantau CPU *usage*. Diagram blok alur pengujian akan digambarkan pada Gambar 3.8.



Gambar 3.8 Diagram blok alur pengujian.