# BAB 2
# DASAR TEORI

## 2.1. Kajian Pustaka

Ilham Reza Wijaya pada penelitiannya di tahun 2019, melakukan analisis kinerja *load balancing* menggunakan algoritma *dynamic ratio* dan membandingkannya dengan algoritma *round robin* pada *load balancer F5 BIG-IP 13* dengan empat parameter uji yaitu *throughput*, *response time*, *error*, dan *CPU utilization*. dari hasil pengujian menunjukan dengan menggunakan 3 *server* tidak terjadi *overload* pada *server* pada saat pengujian algoritma *dynamic ratio* maupun *round robin*. Nilai rataan pada algoritma *dynamic ratio* sebesar 57,83 KB/s dan pada *round robin* sebesar 55,27 KB/s, nilai rataan *response time* pada algoritma *dynamic ratio* sebesar 2,64 detik dan *round robin* sebesar 2,67 detik, nilai *error* pada algortima *dynamic ratio* sebesar 0,0% dan *round robin* sebesar 0,9% sedangkan nilai *CPU utilization* pada algoritma *dynamic ratio* sebesar 93,5% dan algoritma *round robin* sebesar 93,0%. Nilai *Fairness Index* pada algoritma *dynamic ratio* tidak mencapai angka 1 sedangkan pada algoritma *round robin* dapat mencapai angka 1.

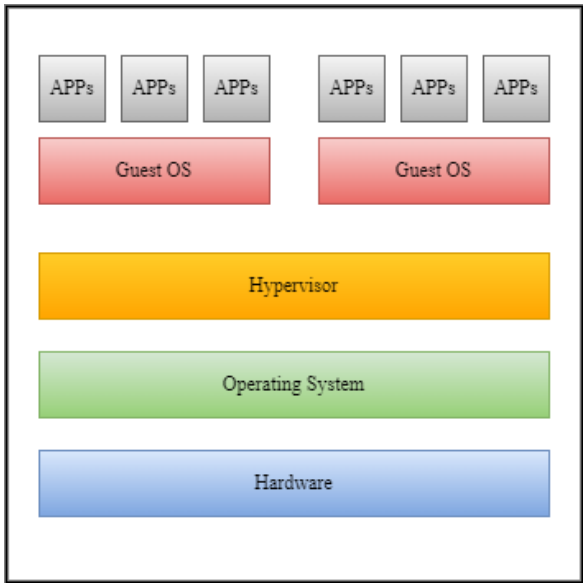
Pada penelitian Rifaldi Adi Putra dan Helmi Faisal Lutfi di tahun 2019, melakukan implementasi *load balancing* menggunakan algoritma *round robin* pada *load balancer F5 BIG-IP LTM* dengan parameter uji adalah *response time*. Hasil penelitian tersebut menunjukkan hasil yang baik karena *availability* yang tetap terjaga tanpa terjadi *overload* pada ketiga *server* dalam pengujian beban *traffic* 10 *request*, 100 *request*, dan 1000 *request*.

Arip Solehudin pada penelitiannya di tahun 2020, melakukan penelitian dengan membandingkan algoritma *round robin* dan *least connection* dengan *load balancer HAProxy* pada *Web Server* menggunakan tiga parameter uji yaitu *throughput*, *response time*, dan *CPU utilization*. Pada penelitian tersebut mendapatkan hasil parameter *throughput* kedua algoritma memperoleh nilai rata-rata yang sama yaitu 99,5 Kb / *second*. Pada pengujian parameter *response time* algoritma *least connection* lebih

unggul yaitu 6,9 ms dibandingkan dengan *round robin* yang memiliki nilai rata-rata sebesar 7,2 ms, namun pada pengujian parameter *CPU utilization* algoritma *round robin* bernilai 23,7% lebih unggul dibandingkan algoritma *least connection* yaitu 24,3%.

## 2.2 *Virtual Machine*

*Virtual Machine* merupakan sebuah sistem operasi atau aplikasi yang di *install* pada *hypervisor* dan memiliki fungsi layaknya perangkat fisik (*hardware*) atau bisa juga disebut sebagai duplikat dari komputer asli. *Software hypervisor* digunakan untuk membuat dan mengatur *virtual machine*. Selain itu, *hypervisor* juga dapat berperan sebagai agen penghubung antara *virtual machine* dengan perangkat fisik. *Hypervisor* dibagi menjadi dua jenis yaitu *hypervisor* tipe 1 dan 2, di mana tipe 1 berjalan langsung diatas perangkat keras sedangkan *hypervisor* tipe 2 berjalan diatas *host OS*. Contoh dari tipe 2 adalah *oracle virtualbox* dan *vmware workstation* sedangkan untuk tipe 1 adalah *vsphere* dan *citrix xen server*. *Hypervisor* dapat menjalankan perangkat lunak apa pun yang berjalan pada perangkat keras *bare metal* sementara menyediakan isolasi dari perangkat keras yang sebenarnya [2]. Ilustrasi skema *virtual machine* terdapat pada gambar 2.1.




**Gambar 2.1. Arsitektur *Virtual Machine***

Keuntungan dalam menggunakan Virtualisasi adalah sebagai berikut :

1.) *Portability*

Kemampuan untuk mempunyai *platform hardware* yang konsisten, bahkan ketika *hardware* yang asli berasal dari manufaktur yang berbeda.

2.) *Manageability*

*Virtual environment* bisa dikelola dengan mudah dan menawarkan akses ke *virtual hardware*.

3.) *Efficiency*

Saat diimplementasi dengan benar, *server virtualization* memungkinkan *hardware* fisik digunakan secara lebih efisien, sehingga memungkinkan penggunaan dari *hardware* menjadi lebih tinggi.

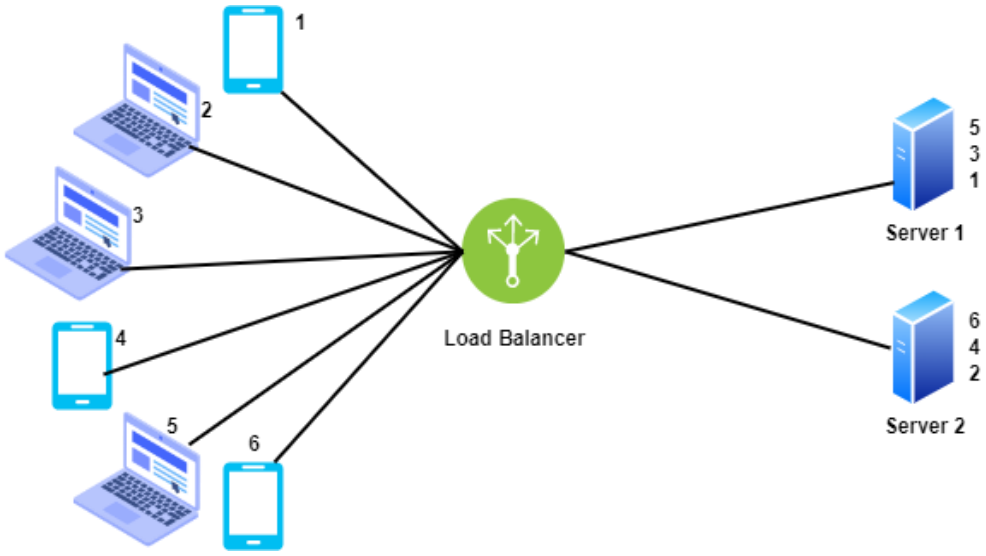
## 2.2. *Load Balancing*

*Load balancing* merupakan teknik pendistribusian data pada dua jalur atau lebih secara seimbang dengan tujuan untuk mendapatkan *traffic* yang lebih optimal, mengurangi waktu tanggap, dan mencegah terjadinya penumpukan data pada *server* [3]. Proses pembagian tersebut dimulai ketika *load balancer* menerima *request* layanan pengguna dan selanjutnya *request* tersebut akan didistribusikan menuju *node-node server* yang telah terdaftar dalam sistem *load balancer*. Mekanisme pendistribusian paket yang dilakukan oleh *load balancer* telah diatur sesuai algoritma yang ditentukan sebelumnya. Penerapan mekanisme yang sistematis tersebut memungkinkan pembagian *request* yang lebih efisien sehigga unjuk kerja *resource* pada jaringan lebih stabil dan dapat meminimalisir terjadinya *overload*. *Load balancing* perlu diterapkan pada sebuah jaringan *server* yang memiliki potensi jumlah *client* melebihi dari kapasitas serta memiliki waktu terhubung dengan *server* yang bervariasi. *Load balancing* tidak hanya membagi beban pada pada sisi *server*, tetapi juga memperhatikan penggunaan sumber daya yang digunakan seperti memori, CPU, dan *resource* lainnya, agar lebih efisien dan optimal [4].

### 2.2.1. Algoritma *Round Robin*

Algoritma *round robin* adalah salah satu algoritma yang umum digunakan dalam *load balancing*. Algoritma ini berjalan dengan cara membagi beban *traffic* secara bergiliran dan berurutan dengan porsi yang sama rata dari satu *server* ke *server* lain. Konsep dasar dari algoritma *round robin* adalah dengan menggunakan *time sharing*. Setiap proses mendapatkan waktu CPU yang disebut dengan waktu *quantum* untuk membatasi waktu prosesnya, biasanya 1-100ms. Algoritma ini memproses antrian secara bergiliran [5].

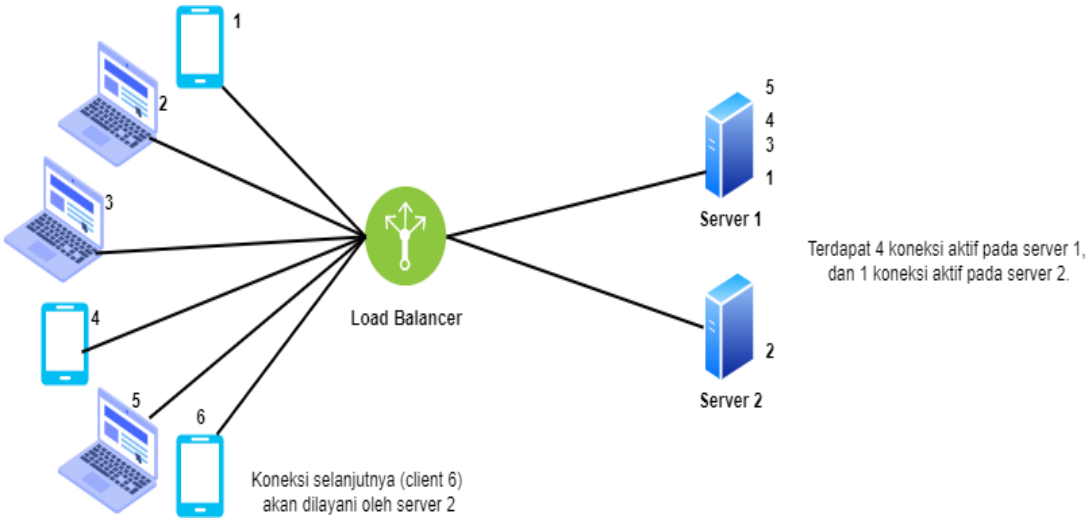
Proses akan mendapat jatah sebesar waktu *quantum*. Jika waktu *quantum*nya habis atau proses sudah selesai, CPU akan dialokasikan ke proses selanjutnya. Sebaliknya, jika suatu proses memiliki CPU *burst* yang lebih besar dibandingkan dengan waktu *quantum*, maka proses tersebut akan dihentikan sementara jika sudah mencapai waktu *quantum*, dan selanjutnya mengantri kembali pada posisi akhir dari *ready queue*, CPU kemudian menjalankan proses berikutnya [5]. Proses pembagian *traffic* dijelaskan pada gambar 2.2.




**Gambar 2.2. Mekanisme Pembagian *Request* Pada Algoritma *Round Robin***

### 2.2.2. Algoritma *Least Connection*

Algoritma *least connection* merupakan algoritma yang bekerja dengan cara mendistribusikan lebih banyak *request* ke *server* asli berdasarkan koneksi aktif yang lebih sedikit. *Server* dengan pelayanan koneksi yang paling sedikit akan diberikan beban yang berikutnya akan masuk. Penggunaan algorima *least connection* sangat sesuai diterapkan pada *cluster* dengan lingkungan yang dinamis dengan *session* yang berubah-ubah [6]. Gambar 2.3 mengilustrasikan di mana *server* 1 terindikasi memiliki antrian lebih padat dibandingkan *server* 2 sehingga *request client* selanjutnya (*client* 6) akan dilayani oleh *server* 2. Kemungkinan hal tersebut terjadi dikarenakan *client* yang tersambung ke *server* 2 tetap terus terhubung lebih lama dibandingkan dengan yang terhubung ke *server* 1. Hal tersebut dapat menyebabkan jumlah total koneksi di *server* 2 menumpuk, sementara mereka pada *server* 1 (dengan *client* terhubung dan terputus selama waktu yang lebih pendek) hampir akan tetap sama.




**Gambar 2.3. Mekanisme Pembagian *Request* Pada Algoritma *Least Connection***

## 2.3. *F5 BIG-IP LTM*

*F5* merupakan sebuah *brand* yang didirikan pada tahun 1996 dan saat ini sedang mendominasi market *load balancer* dan juga *application delivery controller* dengan salah satu produk unggulannya adalah *F5 Big-IP Local Traffic Manager* (LTM)

yang merupakan perangkat *load balancer* dengan keunggulan manajemen *traffic* pada jaringan lokal. Produk *F5 BIG-IP* memiliki berbagai versi, salah satunya versi *virtual edition* series terbaru yaitu 17.0.X yang dapat dijalankan pada *VMware* dengan *file extension* (*.ova*) serta spesifikasi *throughput* maksimal 40 Gbps. *Load balancer* F5 BIG-IP LTM memiliki keunggulan tersendiri dengan sejumlah besar fitur tambahan yang dirancang untuk lebih memudahkan *operator* dalam mengontrol *traffic* jaringan, memilih tujuan *traffic* dengan benar berdasarkan peforma *server*, serta terdapat fitur keamanan. Keunggulan dari *F5 Big-IP LTM* adalah *Full Proxy*, *Blazing fas SSL*, *TCP Optimization*, *Performance Optimization*, *Programmability*, *Scale*, dan *Speed* [7].

**Gambar 2.4**. ***Load Balancer F5 BIG-IP LTM*** **[8]**

## 2.4. *Web Server*

*Web server* adalah *server* berbetuk *software* yang melayani HTTP *request* dari *client* melalui *web browser* dan mengirimkan kode dinamis ke *server* aplikasi. *Server* ini akan memproses kode dinamis menjadi kode statis dalam suatu halaman statis yang selanjutnya dikirimkan ke *browser* oleh *web server*. Penggunaan yang paling umum adalah digunakan sebagai tempat untuk situs *website* namun pada prakteknya penggunaan *web server* diperluas sebagai tempat penyimpanan data atau menjalankan sejumlah aplikasi [9].

Cara kerja *web server* adalah ketika *client* melakukan *request* halaman *website*, maka *client* akan mengirimkan HTTP *request* dan dikirim ke alamat *web server*. Kemudian *web server* melakukan pengecekan keamanan sebelum berlanjut

untuk pemrosesan *request* yang dibantu oleh HTTP *server* dan selanjutnya *server* akan membalas dengan mengirimkan HTTP *response* berupa sinyal *synchronize* dan *acknowledge* kepada *client*. Selanjutnya *client* membalas dengan sinyal *acknowledge* yang menandakan sudah siap untuk mengirimkan data [10].

Data akan ditampilkan oleh *browser* sesuai dengan kemampuan *browser*. Apabila data yang dikirim berupa gambar, *browser* yang hanya mampu menampilkan teks (misalnya *lynx*) tidak dapat menampilkan gambar tersebut, atau hanya akan menampilkan alternatifnya saja [11].

## 2.5. *H2load Benchmark*

*H2load* adalah salah satu alat pengujian *web server* baik untuk HTTP/2 maupun HTTP/1 dengan dukungan SSL atau TLS. *H2load* menggunakan *io non-blocking* dalam membuat koneksi bersamaan untuk menargetkan titik akhir GET/POST HTTP, sehingga tidak menyebabkan beban menjadi tinggi pada sistem yang berjalan karena *thread* tunggal dapat membuat ribuan koneksi dalam satu detik [12].

## 2.6.QoS (*Quality of Service*)

*Quality of Service* (QoS) merupakan metode pengukuran tentang seberapa baik jaringan dan merupakan suatu usaha untuk mendefinisikan karakteristik dan sifat dari satu servis. QoS digunakan untuk mengukur sekumpulan atribut kinerja yang telah dispesifikasikan dan diasosiasikan dengan suatu servis [13].

### 2.6.1 *Throughput*

*Thoughtput* merupakan kecepatan transfer data efektif dengan satuan bps (*byte per second*). Perhitungan *throughput* yaitu jumlah total kedatangan paket sukses pada tujuan selama interval waktu tertentu dibandingkan dengan durasi interval waktu tersebut. [13]

$$Throughtput\ (bps) = \frac{Paket\ yang\ diterima\ (bit)}{Durasi\ pengamatan\ (\text{sec})} \quad (2.2)$$

Dapat diartikan jika nilai yang dihasilkan *throughput* semakin besar maka jaringan akan semakin baik. Faktor yang mempengaruhi nilai *throughput* adalah media transmisi yang digunakan, di mana setiap media transmisi memiliki kapasitas *bandwidth* yang berbeda. Standarisasi *throughput* berdasarkan TIPHON TR 101 309 (1999) diklasifikasikan pada tabel 2.2 [14].

**Tabel 2.1. Klasifikasi Standar *Thoughput***

| Kategori | Nilai Throughput |
|---|---|
| Sangat Baik | >2,1 Mbps |
| Baik | 1200 Kbps – 2,1 Mbps |
| Cukup | 700 – 1200 Kbps |
| Kurang Baik | 338 – 700 Kbps |
| Buruk | 0 – 338 Kbps |

### 2.6.2 *Delay*

*Delay* merupakan waktu yang dibutuhkan untuk menempuh jarak dari asal ke tujuan suatu paket. Satuan dari *delay* adalah *millisecond* (ms). *Delay* dipengaruhi oleh beberapa faktor yaitu jarak, media fisik dan waktu proses yang lama.

$$Delay = \frac{Panjang\ paket}{Bandwidth\ media} \quad (2.3)$$

Standarisasi *delay* berdasarkan TIPHON TR 101 309 (1999) diklasifikasikan pada tabel 2.3 [14].

**Tabel 2.2. Klasifikasi Standar *Delay***

| Kategori | Nilai *Delay* (ms) |
|---|---|
| Sangat Baik | ≤ 150 |
| Baik | 150 – 300 |

| | |
|---|---|
| Sedang | 300 – 450 |
| Kurang | > 400 |

Dapat diartikan bahwa, jika nilai yang dihasilkan *delay* semakin kecil maka kinerja dalam sebuah jaringan semakin baik.

## 2.7.*Response Time*

*Response time* merupakan waktu yang dibutuhkan oleh *server* dalam menangani *request* dari *client*. Nama lain *response time* adalah *latency* yang merupakan total waktu tunggu dan waktu menjawab *request* [15]. Tujuan utama *load balancing* adalah untuk menyediakan *response time* terbaik kepada *client* dengan cara menyebarkan beban ke *server* dalam *cluster*. Jika *response time server* tinggi, akan ada *request* yang menunggu untuk ditangani *server*. Jika *response time server* rendah, *request* yang menunggu untuk ditangani juga rendah karena *server* melayani *request* dengan *rate* tinggi. Jika tiap *server* mempunyai *response time* yang berbeda-beda, *request* akan dikirimkan ke *server* dengan beban yang lebih sedikit [16]. Parameter *response time* berdasarkan riset *human factors pioneers* yang dilakukan oleh Don Norman dan Jakob Nielsen pada tahun 1993, nilai 0,1 *second* termasuk kategori sangat baik, sedangkan 1 *second* termasuk kategori cukup, dan 10 *second* tergolong kategori buruk [17].