BAB II
DASAR TEORI

2.1 KAJIAN PUSTAKA

Pada tahun 2019, Apriliansyah, dkk. [3] menganalisis implementasi *load balancer* pada *web server* menggunakan algoritma *least connection* dan *round robin*. Dalam penelitian tersebut, dilakukan pengujian transfer *rate* waktu akses yang dilakukan dengan jumlah 4000, 8000, 12000, dan 16000 *requests*, untuk 200 klien. Parameter uji yang diukur adalah *throughput* dan *response time*. Dari hasil pengujian didapatkan algoritma *least connection* menghasilkan *throughput* yang stabil pada kecepatan 17 Mbps, dimana pada algoritma *round robin* hasil *throughput* didapatkan antara 16 Mbps dan 17 Mbps. Nilai *response time* pada algoritma *round robin* lebih tinggi pada 140 ms, karena pada algoritma *round robin* membagi beban pada setiap *server* sehingga antrian menjadi lebih besar jika permintaan meningkat. *Response time* pada algoritma *least connection* memiliki stabilitas dan *response time* lebih rendah pada 116 ms, karena pada algoritma ini akan menentukan *server* mana yang memiliki *request* paling rendah.

Pada tahun 2019, Wibowo, dkk. [5] menganalisis QoS pada *load balancing* dengan algoritma *least connection*. Hasil penelitian ini menunjukkan bahwa *throughput* pada *least connection* termasuk sangat bagus, pembagian beban kerja ke *server* juga merata dengan *response time* yaitu *processing* time 75 *second* dan minimal *processing time* sebesar 22 *second*. Algoritma *least connection* memiliki pembagian beban jaringan yang baik, dilihat dari kecepatan rata rata permintaan yang diselesaikan oleh setiap *server* dapat diselesaikan dalam waktu yang relatif singkat (sekitar 40ms).

Pada tahun 2017, Zakia dan Yedder. [6] menganalisis penerapan *load balancing* pada *Software Define Network* (SDN) berbasis data *center networks*. Pada penelitian ini menguji beberapa parameter yaitu *throughput, delay, packet loss*. Hasil pengujian menunjukkan bahwa *packet loss* masih dalam kategori bagus, dikarenakan paket yang hilang hanya sebesar 0 – 3%.

Pada tahun 2019, Kumar [7] menganalisis penerapan *Moodle* berbasis *cloud computing* untuk universitas. Penelitian ini berdasarkan tiga skenario dengan

jumlah pengguna yaitu 200 pengguna, 2000 pengguna dan 10.000 pengguna. Penelitian ini akan menunjukkan waktu yang dibutuhkan pengguna dalam mengakses LMS. Terdapat lebih dari tiga juta jumlah akses ke halaman LMS, lama durasi kunjungan dari tiga menit hingga 30 menit. Ini menunjukkan adanya peningkatan keterlibatan pengguna dalam alat belajar dan mengajar yang menggunakan LMS.

Pada tahun 2019, Sadikin, dkk. [8] melakukan penerapan *load balancing* untuk mengatasi masalah kinerja pada sistem *e-learning*. Pemantauan menggunakan robot untuk memantau memantau kinerja *e-learning* seperti *date-time*, *port-accessed*, jumlah akses pengguna, dan ringkasan *up-time*. Hasil pengujian ini menunjukkan jumlah maksimum kunjungan pengguna per menit adalah 1.528 pengguna. Kecepatan akses menjadi lebih baik, pengujian selama 2 bulan waktu aktif pada sistem *e-learning* menunjukkan di angka puncak 100%.

Tabel 2.1 Rangkuman keterkaitan penelitian dengan kajian Pustaka.

No	Peneliti	Tahun	Tujuan	Metode	Parameter	Hasil
1	Apriliansyah, dkk.	2019	Implementasi *load balancing* pada *web server*.	*Load balancing* dengan algoritma *least connection* dan *round robin*.	*Throughput*, *response time*, CPU *usage*.	Hasil *throughput* pada *least connection* stabil pada 16 Mbps - 17 Mbps. *Response time* lebih rendah di 116 ms.
2	Wibowo, dkk.	2019	Analisis QoS pada *load balancing* dengan algoritma *least connection*.	*Load balancing* dengan algoritma *least connection*.	*Response time* dan CPU *usage*.	Hasil throughput pada least connection termasuk sangat bagus, response time yaitu processing time 75 s dan minimal processing time sebesar 22s.

No	Peneliti	Tahun	Tujuan	Metode	Parameter	Hasil
3	Zakia dan Yedder.	2017	Analisis penerapan *load balancing* pada *software define network*.	*Load balancing* pada *software define network*.	*Throughput* dan *packet loss*.	Hasil pengujian menunjukkan *packet loss* masih dalam kategori yang bagus karena paket yang hilang sebesar 0%.
4	Kumar.	2019	Analisis penerapan *Moodle* berbasis *cloud computing* untuk universitas.	LMS *Moodle* berbasis *cloud computing*.		Lebih dari tiga juta jumlah akses LMS, dengan waktu kunjungan 3-30 menit. Ini menunjukkan adanya peningkatan kunjungan ke LMS.
5	Sadikin, dkk.	2019	Implementasi *load balancing* untuk mengatasi masalah kinerja *e-learning*.	*Load balancing* pada *e-learning*.		Jumlah maksimum kunjungan per menit 1.528 pengguna. Kecepatan akses lebih baik dan waktu aktif *e-learning* 100% dalam dua bulan

2.2 *LEARNING MANAGEMENT SYSTEM*

Learning Management System (LMS) adalah sistem yang dapat mengelola pembelajaran [9]. Untuk menyediakan sistem pembelajaran berbasis internet yang dapat membantu peserta didik, diperlukan adanya sistem yang dapat menyediakan kursus untuk membantu guru dan peserta didik berpartisipasi dalam pembelajaran. Fungsi utama LMS adalah media belajar, manajemen kehadiran dan absensi, yang semuanya diperlukan untuk pembelajaran daring.

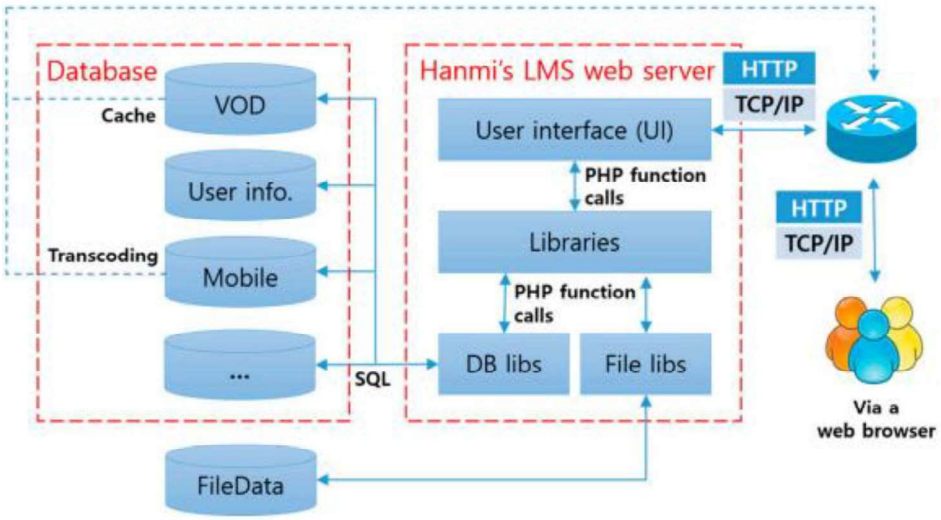
Fitur-fitur yang disediakan dalam LMS untuk Pendidikan sebagai berikut.

1) Manajemen akses pengguna.
2) Manajemen mata kuliah.
3) Manajemen bahan ajar (sumber daya).
4) Manajemen visualisasi sehingga dapat digunakan melalui situs internet.

LMS tersedia dengan basis web, dikembangan dengan berbagai platform seperti JAVA, *Microsoft*.NET atau PHP. LMS menggunakan beberapa *database* seperti MySQL, *Microsoft* SQL *server* atau *Oracle* sebagai *backend* dalam pengembangannya [10].

Fungsi-fungsi LMS agar dapat disediakan secara online membutuhkan perangkat jaringan seperti *server* yang akan merespon permintaan pengguna untuk mengakses *web* LMS, *database* untuk menyimpan data pembelajaran. Perangkat jaringan yang digunakan dan dikonfigurasi sesuai kebutuhan penyedia LMS [10].

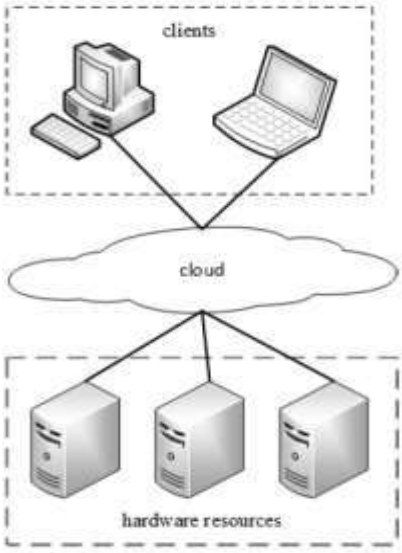
Chamilo LMS adalah platform *e-learning* yang lengkap [10]. *Chamilo* LMS dapat dipasang dalam waktu singkat, dan digunakan oleh guru dan siswa . *Chamilo* berbasis *open source* sehingga mudah diimplementasikan dan mudah untuk mempersiapkan pembelajaran [10].



Gambar 2.1 LMS *system architecture* [8].

2.3 *CLOUD COMPUTING*

Cloud Computing adalah teknologi komputasi baru yang pada dasarnya menggunakan *server* virtual yang disediakan melalui internet [11]. Konsep *cloud computing* adalah sistem yang dapat menambahkan kemampuan infrastruktur-nya secara dinamis, tanpa perlu mengeluarkan uang untuk membangun infrastruktur baru, tanpa perlu menambah dan memberikan pelatihan kepada pengelola infrastruktur-nya, tanpa perlu membeli lisensi perangkat lunak. Infrastruktur *cloud computing* sangat andal dan dibangun dengan teknologi virtualisasi [11]. Sumber daya seperti penyimpanan, komputasi atau jaringan, disediakan kepada pengguna kedalam layanan berupa infrastruktur, *platform* atau perangkat lunak. Infrastruktur *cloud computing* sangat andal dan dibangun dengan teknologi virtualisasi. Model *cloud computing* yang diterapkan pada internal perusahaan disebut *private cloud* dan *cloud computing* yang disediakan ke publik oleh *provider cloud* disebut *public cloud* [12].



Gambar 2.2 Topologi dalam penggunaan *cloud computing* [12].

Terdapat lima elemen mendasar pada *cloud computing* sebagai berikut [13].

1) *On-demand self-service*

   Pengguna *cloud computing* dapat menentukan sumber daya komputasi yang dibutuhkan seperti *database*, jaringan, CPU, *storage*, dll secara otomatis atau melayani diri sendiri.

2) *Broad network access*

Sumber daya *cloud computing* disebarkan melalui jaringan internet serta dapat diakses oleh pengguna *cloud computing* di mana saja dan menggunakan platform apa saja seperti laptop, handphone, dll. Sumber daya dapat diakses melalui sebuah situs.

3) *Resource pooling*

Penyedia layanan mengumpulkan sumber daya komputasi yang dibutuhkan pengguna dengan model *multi-tenant*. Model ini bisa berupa *server* fisik atau virtual. Pada *cloud computing* sumber daya berada pada *data center*, dan sumber daya disebarkan kepada pengguna di tempat yang berbeda-beda.

4) *Rapid elasticity*

Sumber daya komputasi yang digunakan pengguna dapat ditambahkan apabila kebutuhan meningkat, dan dapat dilepaskan ketika tidak lagi dibutuhkan. Oleh karena itu, penyediaan sumber daya komputasi jadi tidak terbatas dalam menangani permintaan yang terus meningkat.

5) *Measured service*

Penggunaan sumber daya komputasi perlu adanya monitoring untuk memantau penggunaan. Sistem ini dapat melihat berapa sumber daya komputasi yang terpakai, banyaknya layanan yang dipakai dan biaya yang dikeluarkan.

Model layanan *cloud computing* terdiri dari beberapa model sebagai berikut [13].

1) *Software as a Service* (SaaS)

Pada model layanan ini pengguna dapat mengakses layanan melalui *web* ataupun aplikasi. Pengguna *cloud computing* tidak memiliki kendali terhadap infrastruktur *cloud computing*. Pengembangan kode dalam menjalankan aplikasi dilakukan pada lingkungan *cloud* SaaS.

2) *Platform as a Service* (PaaS)

Pada model layanan ini, memungkin pengguna untuk mengembangkan aplikasi langsung pada *cloud* PaaS. Sehingga pengguna tidak perlu memikirkan *server* atau infrastruktur, hanya perlu mengembangkan aplikasi. Model *cloud* PaaS adalah pengembangan dari platform yang mendukung "*Software Lifecycle*" sehingga memungkinkan pengguna untuk mengembangkan aplikasi pada *cloud* PaaS.

3) *Infrastructure as a Service* (IaaS)

Pada model layanan ini pengguna dapat menentukan infrastruktur seperti *storage*, jaringan, *database*, dll sesuai kebutuhannya. IaaS menggunakan teknologi virtualisasi untuk memenuhi kebutuhan infrastruktur pengguna. Pengguna dapat memilih infrastruktur berupa VM yang telah diisolasi dari *data center*, sehingga pengguna dapat memilih VM, *Operating System* (OS), *storage*, *database*, dll sesuai kebutuhan.

Terdapat lima model penyebaran *cloud computing* yang telah didefinisikan komunitas *cloud computing* sebagai berikut [13].

1) *Private cloud*

Pada model penyebaran ini infrastruktur dioperasikan untuk keperluan perusahaan serta dikelola oleh perusahaan itu sendiri atau pihak ketiga. Tujuan memanfaatkan *private cloud* untuk memanfaat sumber daya internal yang ada dan aspek keamanan dan kerahasiaan data perusahaan. Akademisi sering memanfaatkan *private cloud* sebagai penelitian dan pengajaran.

2) *Community cloud*

Beberapa perusahaan membangun infrastruktur *cloud computing* secara bersama-sama untuk digunakan bersama. *Community cloud* akan meningkatkan skalabilitas ekonomi. Infrastruktur dapat dibangun oleh pihak ketiga atau perusahaan yang mampu membangun infrastruktur *cloud computing*.

3) *Public cloud*

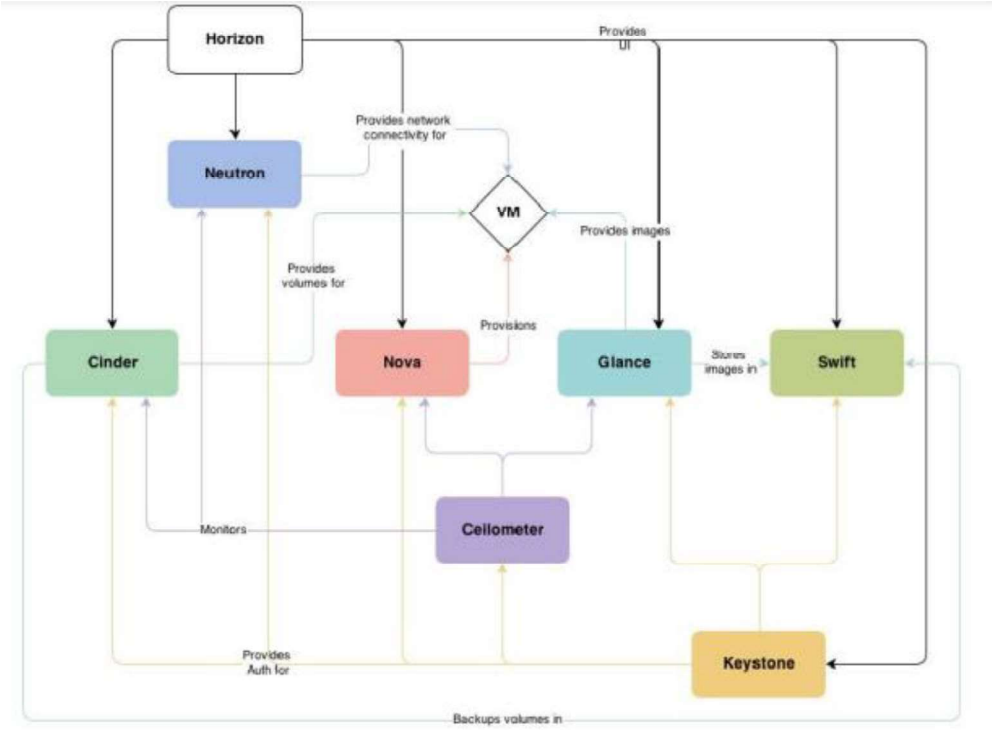
Model ini adalah model penyebaran yang dominan digunakan oleh pengguna *cloud computing*. *Public cloud* digunakan untuk pengguna *cloud computing* dan *cloud provider* memiliki hak penuh atas *cloud computing* serta bertanggung jawab atas *data center* yang diperlukan sebagai sumber daya komputasi. *Public cloud* memiliki kebijakan terhadap profit, biaya. Beberapa yang populer seperti *Amazon Web Service* (AWS)*, Microsoft Azure, Google Cloud Platform* (GCP).

4) *Hybrid cloud*

Model ini merupakan kombinasi dari dua atau lebih model penyebaran *cloud computing private*, *public*, atau *community*. Model ini digunakan perusahaan untuk mengoptimalkan sumber daya perusahaan dan meningkatkan kompetensi mereka.

2.4 *OPENSTACK*

Openstack dirancang sebagai sebuah *software* yang mengoperasikan *cloud computing* dan dapat diskalakan. *Openstack* merupakan layanan dari *Infrastructure as a service*" (IaaS). Layanan ini berjalan melalui antarmuka publik atau *Application Programming Interfaces* (API), antarmuka ini dapat diakses oleh pengguna akhir *Cloud* [14]. *Openstack* merupakan *software open source*, memiliki keandalan dan ketahanan sebagai IaaS [14]. *Openstack* dapat menjadi solusi untuk infrastruktur *cloud computing*, baik *public cloud* ataupun *private cloud*. *Openstack* merupakan arsitektur modular dan dapat dikonfigurasi, sehingga memungkinkan untuk disesuaikan dengan sumber daya perangkat keras perusahaan. Ini memudahkan perusahaan memilih layanan sesuai dengan kebutuhan mengenai komputasi, jaringan dan penyimpanan [12].



Gambar 2.3 Konsep arsitektur *Openstack* [12].

1) *Nova*

Dikenal sebagai *openstack compute*, Nova menyediakan *server* virtual yang terintegrasi dengan *hypervisor*. *Nova* merupakan perangkat lunak yang menyediakan sumber daya *cloud* dan mengatur *server* virtual yang berjalan.

2) *Glance*

Merupakan layanan *image openstack*, pencarian dan pengambilan sistem pada *image server* virtual. *Glance* layanan untuk mengambil *image* yang akan digunakan pada *server* virtual.

3) *Neutron*

Layanan ini menyediakan konektivitas jaringan antar muka perangkat yang dikelola oleh layanan yang lain. *Neutron* memiliki kemampuan untuk mengatur konfigurasi *Dynamic Host Configuration Protocol* (DHCP), *static* IP, *virtual area network*. Arsitektur ini memungkinkan pengguna untuk menentukan kerangka kerja seperti *load balancing* dan *Virtual Private Network* (VPN).

4) *Swift*

Swift merupakan *openstack object storage*. *Swift* menyediakan penyimpanan yang sangat tersedia, tempat dimana pengguna menyimpan dan mengambil *file*.

5) *Cinder*

Cinder merupakan *openstack block storage*. *Cinder* bekerja dengan *swift*, layanan ini dapat digunakan sebagai *backup* volume VMs.

6) *Keystone*

Keystone adalah identitas *Openstack* layanan ini menentukan kebijakan *Openstack*, katalog, token dan otentikasi, mengatur untuk berintegrasi dengan pengguna dan layanan. Akibat tidak adanya *Keystone*, kepatuhan antar layanan tidak terjadi. Sehingga *cloud computing* akan tidak tersedia untuk pengguna yang membutuhkan solusi *cloud computing*.

7) *Horizon*

Horizon dikenal sebagai *dashboard Openstack*, layanan ini merupakan aplikasi *web* yang menyediakan antarmuka pengguna untuk menyiapkan dan mengelola infrastruktur *cloud computing*. Layanan ini beriteraksi dengan API layanan lainnya.

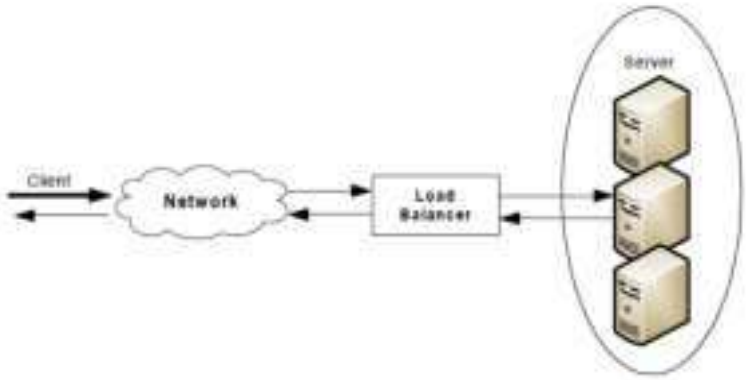
8) *Ceilometer*

Merupakan layanan baru pada *Openstack* yang menyediakan data pengukuran biaya CPU, jaringan, menyediakan kontak untuk sistem penagihan penggunaan [12].

2.5 *LOAD BALANCING*

Load balancing (LB) adalah proses penyesuaian beban kerja pada *server*, LB akan memastikan setiap *server* tidak mengalami kelebihan beban kerja [15]. Ketika jumlah beban kerja berlebih, *server* akan *down* dan tidak ada lagi *server* yang dapat melayani permintaan. *Load balancing* dapat mempercepat kinerja *server* seperti waktu respons, waktu eksekusi, dan kestabilan sistem. Penentuan sumber daya dibutuhkan untuk menentukan status *server* apakah seimbang, tidak seimbang atau kekurangan beban. Setelah sumber daya telah ditentukan, maka tugas dialokasikan kepada sumber daya pada *server* dengan algoritma penjadwalan [2]. Mengurangi konsumsi energi dan menghindari *bottlenecks*, kebutuhan sumber daya, dan memenuhi parameter QoS dalam meningkatkan performa *load balancing*. Hal ini merupakan tujuan lain dari penggunaan *load balancing* [15].

Masalah ketidakseimbangan beban adalah hal yang tidak diinginkan oleh *Cloud Service Provider*, hal ini dapat menurunkan kinerja dan kemanjuran sumber daya komputasi bersama dengan jaminan QoS pada *Service Level Agreement* (SLA) antara konsumen dan penyedia. *Load balancing* dalam *cloud computing* dapat berupa dilakukan pada level *server* fisik atau level VM. Pemanfaatan sumber daya VM dan ketika sekelompok tugas tiba di VM, sumber daya habis yang berarti tidak ada sumber daya yang sekarang tersedia untuk menangani tambahan permintaan. Ketika situasi seperti itu muncul, VM dikatakan telah memasuki kondisi kelebihan beban. Pada titik waktu ini, tugas akan mengalami *down* atau berakhir dengan jalan buntu tanpa harapan untuk mencapainya. Akibatnya ada kebutuhan untuk memigrasikan tugas ke sumber daya VM yang lain. Proses migrasi beban kerja mencakup tiga langkah dasar: *load balancing* yang memeriksa beban saat ini pada sumber daya, penemuan sumber daya yang sesuai untuk menerima permintaan dan migrasi beban kerja dengan memindahkan tugas tambahan ke sumber daya yang tersedia. Operasi ini dilakukan oleh tiga unit berbeda yang biasa dikenal sebagai *load balancer*, *resource discovery* dan *task migration*.



Gambar 2.4 Skema *load balancing* [5].

Algoritma penjadwalan diklasifikasikan kedalam dua tipe yaitu: *static* dan *dynamic* [16].

1) *Static load balancing algorithms*

   Static load balancing algorithms membagikan beban kerja *server* dengan berdasarkan kemampuan *server* dalam menangani permintaan yang baru masuk. Metode ini hanya berdasarkan informasi sebelumnya tentang kemampuan *server*. Hal ini meliputi kinerja *server*, memori dan kapasitas penyimpanan dan kinerja komunikasi yang sudah diketahui. Metode ini tidak bisa beradaptasi dengan perubahan dinamis, dikarenakan algoritma ini tidak memperhitungkan perubahan dinamis pada saat *run time*.

2) *Dynamic load balancing algorithms*

   Dynamic load balancing algorithms mempertimbangkan kemampuan *server* dan bandwith jaringan dalam membagikan beban kerja. Algoritma ini mengumpulkan informasi setiap *server* pada saat *run time* untuk memproses tugas pembagian beban kerja. Perubahan beban kerja yang terjadi saat pembagian beban kerja, metode ini akan bersifat dinamis dengan membagikan beban kerja kembali ke setiap *server* berdasarkan atribut yang dikumpul dan dihitung.

Algoritma penyeimbangan beban sangat penting untuk meningkatkan kinerja *server*. Ketika permintaan masuk, algoritma memilih *server* yang paling sesuai dan menetapkan permintaan untuk itu. Algoritma penyeimbangan beban bekerja berdasarkan prinsip bahwa dalam situasi apa beban kerja berada ditetapkan, selama

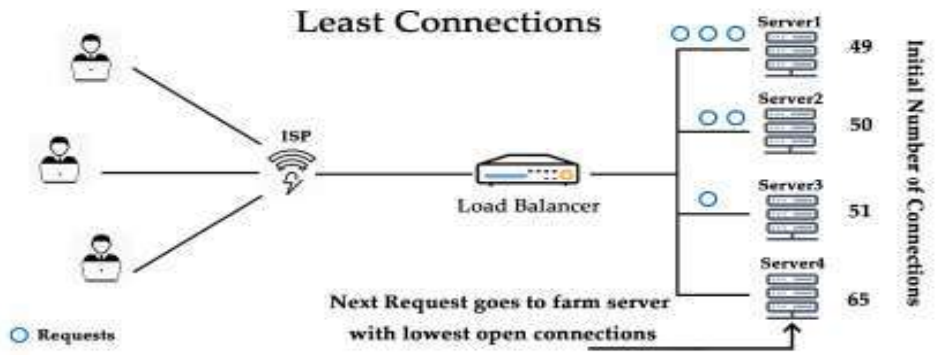
waktu kompilasi atau saat *runtime*. Terdapat beberapa algoritma dalam *load balancing* sebagai berikut.

1) *Round robin*

   Algoritma penjadwalan *round robin* meneruskan setiap permintaan masuk ke *server* berikutnya di daftar. Jadi dalam tiga *cluster server* (*server* A, B dan C) permintaan 1 akan pergi ke *server* A, permintaan 2 akan pergi ke *server* B, permintaan 3 akan pergi ke *server* C, dan permintaan 4 akan pergi ke *server* A, dengan demikian algoritma ini membagikan beban secara merata ke setiap *server*. Algoritma ini membagikan beban ke setiap *server* terlepas dari jumlah koneksi masuk atau waktu respons yang dialami setiap *server*. Algoritma *round robin*, membagikan permintaan yang diterima secara merata dengan cara *round robin*. Ini memperlakukan semua beban *server* sebagai sama terlepas dari jumlah Koneksi. Penjadwalan beban dengan algoritma *round robin* memiliki kelemahan yang akan menyebabkan ketidakseimbangan di antara *server* [17].

2) *Least connection*

   Algoritma penjadwalan *least connection* membagikan beban kerja kepada *server* yang memiliki permintaan paling sedikit [18]. Algoritma ini akan mendistribusikan permintaan ke *server* dan akan memilih *server* mana yang memiliki koneksi paling sedikit. Algoritma ini termasuk ke penjadwalan dinamis, karena akan menghitung koneksi aktif setiap *server* untuk membagikan beban kerja ke *server*. Algoritma ini sangat bagus digunakan untuk membagikan beban kerja ketika permintaan ke *server* terjadi peningkatan permintaan [19]. *Broker* akan menyimpan digit pekerjaan setiap *server*, ketika pekerjaan dikirim jumlah digit akan lebih besar dan jumlah akan dikurangi jika pekerjaan pada *server* telah selesai dieksekusi [20]. Algoritma *least connection* memperkirakan jika kemampuan setiap *server* sama serta membagikan permintaan yang masuk ke setiap *server* dengan koneksi paling kecil. Namun, kinerja *server* kurang bagus saat setiap *server* memiliki kemampuan pemrosesan yang berbeda [17]. Algoritma *least connection* memiliki *throughput* yang lebih tinggi daripada algoritma *round robin*, sehingga algoritma *least connection* disarankan untuk penggunaan yang memerlukan *throughput* tinggi [21].



Gambar 2.5 Konsep algoritma *least connection* [5].

Algoritma *least connection* melihat status beban *real-time* dari *backend node server* dengan melihat nomor koneksi yang tercatat pada *load balancer*, lalu membagikan beban berdasarkan koneksi ke *server* yang paling sedikit. Diberikan asumsi bahwa sebuah sistem memiliki *node server* $S_0, S_1, S_2, \dots, S_{n-1}, C(S_i)$ yang berarti jumlah koneksi pada *server* i dan S_m yang berarti *server* yang ditentukan untuk melayani permintaan yang baru masuk. Algoritma *least connection* dapat dijabarkan sebagai berikut [22].

Input: menetapkan pilihan terakhir yang akan ditetapkan pada node *server* dan total data *server*.

Output: memilih *server* yang akan dibagikan beban.

Least_Connection_Algorithm(node m, *Server*s n)

1) Untuk melintasi *cluster server*
2) Menentukan apakah *server* m telah berfungsi dengan benar, jika tidak berfungsi dengan benar maka kembali ke langkah satu dan beban kerja dipindah ke *server* berikutnya. Jika berfungsi, lanjutkan ke langkah tiga
3) For (melintasi cluster *server* dari *server* m +1);
4) Temukan *server* yang mendapatkan koneksi paling sedikit;
5) Temukan *server* yang tepat, dikembalikan ke S_m;
6) Kembali ke NULL jika tidak mendapatkan *server* yang tepat.

Code Description

```
Least_Connection_Algorithm(node m, Servers n)
{
  for(m=0;m<n;m++)
 {
   if (Sm is alive)
 {
    for(i=m+1;i<n;i++)
        {
         if(C(Si) < C(Sm))
         m=i;
         }
    return Sm
  }
 }
 return NULL;
}
```

Algoritma *least connection* yang merupakan *load balancing* dengan penjadwalan dinamis, dapat membagi koneksi sebagai sebuah unit dan secara dinamis permintaan dibagikan ke *backend server* untuk diproses. Dengan kinerja pembagian beban ini, efisiensi yang dihasilkan oleh *load balancing* akan lebih baik [22].

3) *Least loaded*

Dalam skema ini, operator menetapkan permintaan berikutnya ke *server* yang memiliki beban kerja terendah (beban kerja *server* didefinisikan sebagai jumlah waktu layanan dari semua permintaan yang tertunda *server*). Algoritma ini membutuhkan pengetahuan tentang waktu layanan permintaan klien. Algoritma ini sering tidak mengetahui informasi waktu layanan saat permintaan tiba, dan karenanya sangat sulit untuk menggunakan algoritma ini dalam praktiknya. Namun, operator dapat menggunakan algoritma dasar untuk menetapkan batas pada kinerja [17].

4) *Hashing*

Algoritma penjadwalan statik yang dapat membagikan beban permintaan ke *server* dengan layanan yang diminta. Terdapat suatu tabel *hash* berisi alamat tujuan dari masing-masing *server* beserta layanan yang tersedia pada setiap *server*, atau dengan sistem pembagian dengan *script modulo* [23].

5) *Locality based least connection*

Metode penjadwalan yang akan mengirimkan lebih banyak permintaan kepada *server* yang memiliki koneksi kurang aktif. Algoritma ini akan meneruskan semua permintaan ke *server* yang memiliki koneksi kurang aktif tersebut sampai kapasitasnya terpenuhi [23].

6) *Least traffic*

Algoritma ini membagikan permintaan berdasarkan permintaan yang dilakukan oleh *client*. Setiap permintaan yang masuk *client* untuk koneksi pertama akan *traffic* tersebut diarahkan menuju ke *server* 1, selanjutnya diarahkan ke *server* berdasarkan jalur *traffic* yang rendah dan permintaan ke sekian kali dari *client* akan terus diarahkan berdasarkan perulangan pemilihan *server* dengan *traffic* rendah hingga seluruh permintaan selesai. Pemilihan *server* berdasarkan variabel yang digunakan pada *traffic* dan *path* yang berbeda [24].

7) *Weighted round robin*

Operator secara manual memasukkan bobot atau parameter untuk setiap *server* berdasarkan sumber daya, sehingga operator akan memprioritaskan pembagian kerja *server* berdasarkan beban setiap *server* yang telah ditentukan [25]. Algoritma *weighted round robin* memiliki konsep yang sama dengan algoritma *round robin*, akan tetapi diberikan tambahan kondisi baru yaitu mempertimbangkan kemampuan sumber daya *server* dan memberikan jumlah tugas yang lebih tinggi ke *server* yang mempunyai kapasitas yang lebih tinggi [26].

8) *Weighted least connection*

Merupakan sekumpulan penjadwalan *least connection* dimana dapat ditentukan bobot kinerja pada masing-masing *server*. *Server* dengan nilai bobot yang lebih tinggi akan menerima persentase yang lebih besar dari koneksi-koneksi aktif pada satu waktu. Bobot pada masing-masing *server* dapat ditentukan dan

koneksi jaringan dijadwalkan pada masing masing *server* dengan persentase jumlah koneksi aktif untuk masing-masing *server* sesuai dengan perbandingan bobotnya (bobot awal adalah 1) [23]. *Weighted least-connection* adalah superset dari algoritma *least connection*, di mana operator dapat menetapkan bobot kinerja untuk setiap *server*. *Server* dengan nilai bobot yang lebih tinggi akan menerima persentase lebih besar dari koneksi aktif pada satu waktu. Dalam algoritma *weighted least connection*, sebuah koneksi jaringan baru diberikan ke *server* yang memiliki rasio minimum dari jumlah koneksi aktif saat ini dengan nilai bobotnya. Algoritme *weighted least connection* memiliki daftar *weight* dari *server* dengan jumlah koneksi aktif dari setiap *server* tersebut. Layanan meneruskan koneksi baru ke *server* berdasarkan kombinasi antara nilai *weight* dan banyaknya koneksi aktif dari *server* [27].

2.6 *QUALITY OF SERVICE* (QoS)

QoS adalah memberikan jaminan kemampuan jaringan seperti bandwith, delay, jitter dan kemampuan untuk memenuhi standar SLA antara penyedia layanan dan pengguna. Terdapat beberapa faktor yang dapat menurunkan kualitas QoS seperti redaman, distorsi dan *noise* [25].

1) *Throughput*

Throughput adalah kecepatan transfer data per detik, diukur dalam satuan bps [25]. *Throughput* merupakan total paket data yang diterima saat dikirimkan, dengan interval waktu tertentu dan dibagi oleh durasi interval waktu tersebut. Rumus menghitung *throughput* ditunjukan pada persamaan 2.1.

$$\text{Throughput} = \frac{\text{Paket data diterima}}{\text{Lama pengamatan}} \tag{2.1}$$

2) *Packet loss*

Packet loss adalah parameter yang menghitung total paket yang hilang selama selama pengiriman data dikirimkan [25]. Faktor yang menyebabkan adanya *packet loss* karena adanya *collision* dan *congestion* pada jaringan. Kategori *packet loss* tercantum pada tabel 2.2. Rumus menghitung *packet loss* ditunjukan pada persamaan 2.2.

Tabel 2.2 Kategori *packet loss*.

Kategori	*Packet loss* (%)	*Index*
Sangat Bagus	0	4
Bagus	3	3
Sedang	15	2
Jelek	25	1

$$Packet\ loss = \frac{(Paket\ data\ dikirim - Paket\ data\ diterima) \times 100\%}{Paket\ data\ yang\ dikirim} \quad (2.2)$$

3) *Response time*

Response time adalah waktu yang diperlukan untuk merespons permintaan yang masuk dan mengirimkan kembali kepada *client* [26]. Secara umum kinerja sebuah *server* diharapkan dapat memberikan *response time* yang sependek pendeknya. Tetapi *response time* yang baik memang tidak dapat ditentukan karena ada beberapa aspek yang mempengaruhi antara lain yakni ragam interaksi yang diinginkan dan kinerja dari layanan tersebut.

4) CPU *usage*

CPU *usage* adalah jumlah sumber daya yang diperlukan dalam melaksanakan proses komputasi pada *server* [26]. Peningkatan CPU sejalan dengan banyaknya jumlah permintaan yang dikirimkan, ketika permintaan masuk maka kinerja CPU secara otomatis akan meningkat.