# BAB II
# DASAR TEORI

## 2.1 Kajian Pustaka

Kajian Pustaka pada penelitian ini akan membahas tentang Kajian-kajian sebelumnya yang akan dijadikan acuan dalam penyelidikan ini termasuk dalam kajian pustaka. Selain itu, untuk membandingkan penelitian ini dengan penyelidikan sebelumnya penulis juga akan berbicara tentang perangkat dan prosedur yang digunakan dalam studi sebelumnya.

Menurut penelitian Karim, dkk yang berjudul "Implementasi *Load balancing* Web *Server* Dengan Algoritma *Weighted Least connection* Pada *Software Defined Network*" menggunakan algoritma *Least connection* pada arsitektur *software defined network* yang diterapkan pada *load balancing* di web *server*. Hasil dari penelitian Penerapan *load balancing* dengan algoritma *weighted least connection* dapat membagi beban trafik berdasarkan bobot kinerja dari tiap *server*. Penentuan bobot kinerja ditentukan oleh administrator dengan melihat spesifikasi dari *server* yang digunakan. Pada penelitian ini menggunakan mininet dan *controller* POX sebagai SDN *controller*. Parameter yang digunakan yaitu *connection rate*, *response time*, *throughput*, dan CPU *usage. Tool* yang digunakan untuk pengujian yaitu httperf [10]. Oleh karena itu, pada penelitian ini juga akan menggunakan SDN *Controller* yang sama untuk memenuhi kebutuhan sistem *load balancing*. Parameter penelitian tersebut dapat menjadi pertimbangan dalam penelitian ini untuk digunakan pada tahap pengujian.

Kemudian menurut penelitian Afiani dan Nurwarsito yang berjudul "Implementasi *Load balancing* Web *Server* Pada POX *Controller* Berbasis Penggunaan Memori Dengan Agen Psutil Pada *Software Defined Network*" pada penelitian ini menerapkan metode *load balancing* pada POX *controller* yang berbasis penggunaan memori. Algoritma berbasis penggunaan memori menggunakan agen psutil untuk mengetahui informasi memori pada *server* dan diterapkan dalam POX *controller*. Pada penelitian ini jumlah total *request* koneksi yang diminta oleh setiap *client* sama dengan total jumlah *request* koneksi yang

berhasil direspon oleh *server*. Penggunaan memori berhasil mendistribusikan koneksi dari client ke *server* dengan penggunaan memori terkecil sehingga tidak terdapat nilai *error* dan dapat menghindari *overload* [11]. Oleh karena itu, pada penelitian ini akan menggunakan konsep yang sama dengan controller yang berbeda untuk mengirimkan jumlah koneksi pada *server*.

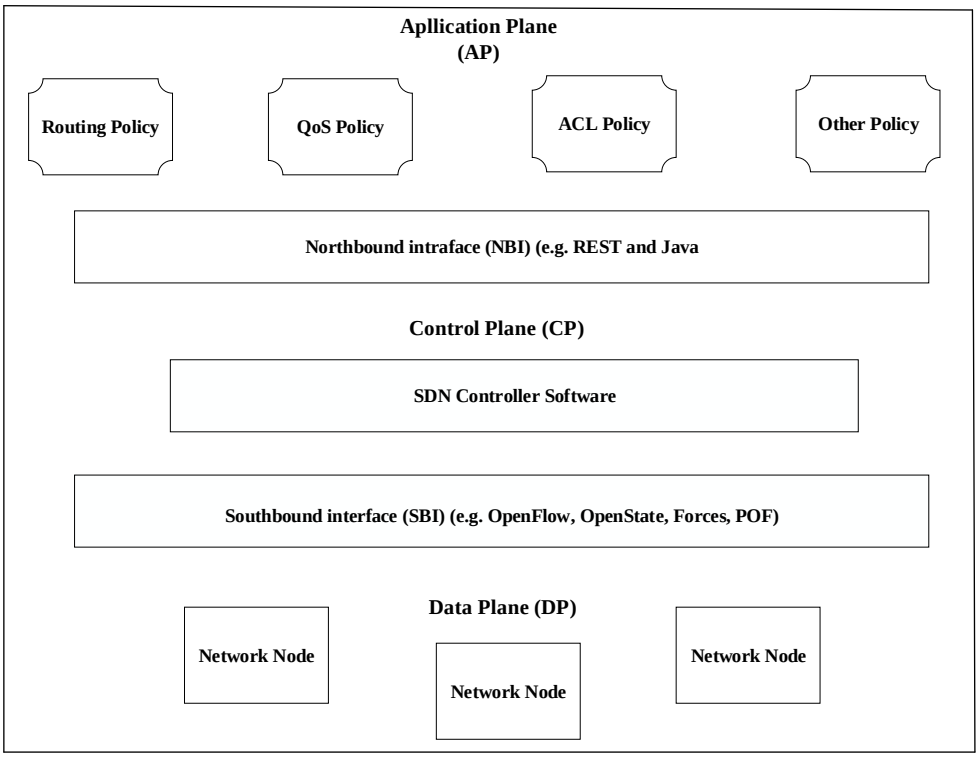
Selanjutnya pada penelitian Sumbayak, dkk yang berjudul "Implementasi *Algortme Weighted Least connection* Berbasis Agen Pada POX *Controller* Untuk *Load balancing* Web *Server* Pada *Software Defined Network*". Penelitian ini menerapkan algoritma *weighted least connection* pada SDN dan juga menggunakan POX *controller* sebagai SDN *controller*. Penelitian tersebut menggunakan algoritma *weighted least connection* (WLCA) untuk mengetahui kinerja dari motode tersebut juga memakai agen. Secara kinerja belum dapat dipastikan karena penelitian tersebut tidak melakukan perbandingan dengan algoritma lain [12]. Oleh karena itu, pada penelitian yang akan dilakukan dapat memperbaiki kekurangan dari penelitian sebelumnya.

## 2.2 Landasan Teori

### 2.2.1 *Software Defined Networking*

Setelah Sun Microsystems menerbitkan Java pada tahun 1995, teknologi Software Defined Network mulai dikembangkan, tetapi pada saat itu tidak cukup dana bagi para peneliti untuk menyelesaikan proyek tersebut. Software Defined Network ini dikembangkan pada tahun 2008 di UC Berkeley dan Stanford University dan kemudian dikembangkan oleh Open Networking Foundation yang didirikan pada tahun 2011. Memperkenalkan teknologi SDN dan OpenFlow dan mulai mempromosikan konsep tersebut [13].

Gambar 2.1 memperihatkan bahwa arsitektur jaringan yang berkembang disebut Software Defined Networking (SDN) memisahkan control plane dari data plane, yang secara langsung diprogram. Arsitektur SDN terdiri dari tiga yaitu application plane (AP), control plane (CP), dan data plane (DP) dan lebih mudah untuk membuat protokol dan aplikasi baru, menampilkan data dan mengumpulkan data dari perangkat middlebox ke dalam kontrol perangkat lunak berkat pemisahan kontrol dan bidang data [14].



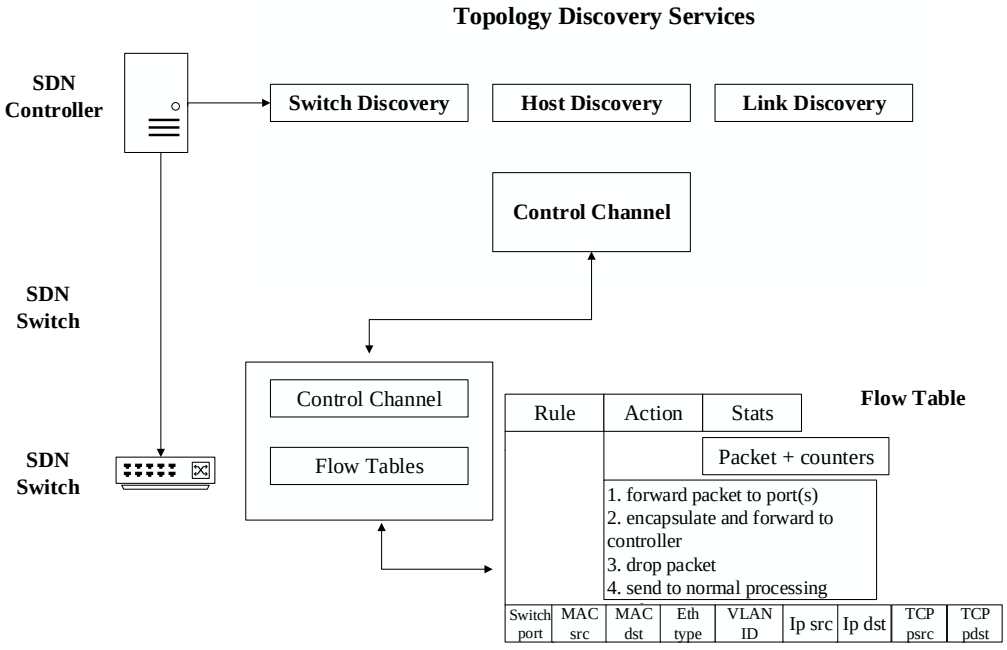

**Gambar 2.1 Arsitektur *software defined network*** [15]**.**

Manfaat memanfaatkan SDN adalah memungkinkan kontrol real-time dari bidang jaringan dengan memperoleh status jaringan secara instan. Hal ini mempengaruhi optimasi kinerja jaringan dan optimasi konfigurasi jaringan.

1. Aplikasi *layer* adalah lapisan yang menampung aplikasi yang merupakan bagian dari jaringan, termasuk aplikasi untuk pemantauan jaringan dan lainnya. *NorthBound Application Programming Interface* adalah sarana yang dengannya lapisan ini dapat berinteraksi dengan *Control layer*.
2. Control *layer*, yang dikenal sebagai Control *layer*, bertugas menginstruksikan *infrastructure layer* pada paket yang masuk ke jaringan. Menggunakan protokol *OpenFlow*, lapisan ini dapat terhubung dengan Lapisan Data *layer*.
3. *infrastructure layer*, yang terdiri dari perangkat keras jaringan, bertanggung jawab untuk mengelola lalu lintas jaringan seperti yang diarahkan oleh *Control layer*. Selain itu, informasi mengenai arahan Control layer akan disimpan di lapisan ini [16].

### 2.2.1.1 *OpenFlow*

Protokol yang disebut *OpenFlow* digunakan dalam SDN dan terletak di antara bidang kontrol dan bidang data. Dalam *switch*, *OpenFlow* dapat mengatur *routing* dan pengiriman paket sehingga *switch* hanya berfungsi untuk meneruskan paket, seperti yang ditujukan pada Gambar 2.2. *OpenFlow* memiliki akses langsung ke dalam kontrol atas bidang penerusan pada perangkat keras seperti *router* dan *switch*. Protokol *Openflow* sudah didukung oleh SDN *Controller* [17]. Selain memberi administrator jaringan akses langsung ke data *traffik* untuk membuat atau meneliti aliran data, *OpenFlow* juga menawarkan protokol terbuka untuk memprogram *flowtable* pada sebagian *switch* dan *router*. Hal ini memungkinkan peneliti untuk mengendalikan aliran data mereka sendiri dengan memilih jalur yang diambil dan diproses oleh paket data yang akan diterima oleh penerima [18].

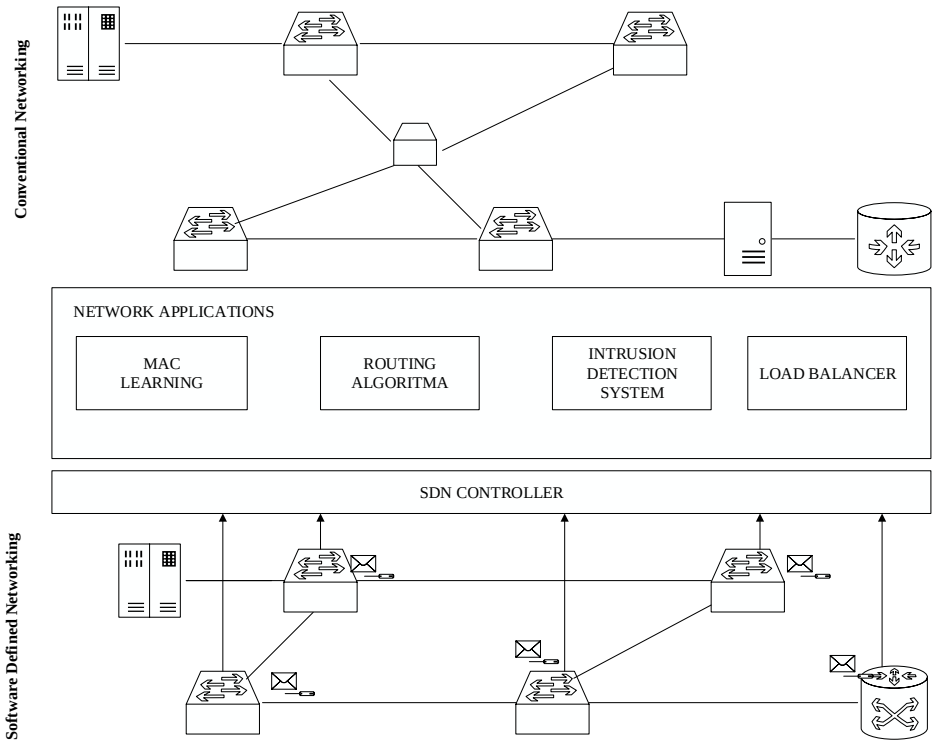



**Gambar 2.2 *Topology discovery processes*** [19]

### 2.2.1.2 SDN *Controller*

*OpenFlow* dikelola oleh perangkat lunak pengontrol yang disebut SDN *Controller*. Pada Gambar 2.3 memanfaatkan SDN memiliki manfaat

memungkinkan kontrol real-time dari bidang jaringan karena akses cepat ke status jaringan. Ini mempengaruhi optimasi pengaturan jaringan dan meningkatkan kinerja jaringan. Salah satu controller arsitektur SDN yang dapat digunakan dalam penelitian ini adalah *Floodlight controller*, yang juga memudahkan *Control Plane* dan *Data Plane* untuk berkomunikasi melalui protokol *OpenFlow* [20].




**Gambar 2.3 SDN Pemisahan Antara *Control Plane* dan *Data Plane*** [13].

### 2.2.1.3 *Floodlight Controller*

*Floodlight* adalah pengontrol *Openflow* berbasis *Java* dengan *lisensi Apache*. Sekelompok pengembang, termasuk beberapa insinyur dari *Big Switch Network* yang mendukung. Semakin banyak *switch*, *router*, *switch virtual*, dan titik akses yang mendukung standar Openflow kompatibel dengan *Floodlight*. Untuk jaringan simulasi, *Floodlight Controller* berfungsi sebagai penyedia jaringan. Perangkat jaringan ini tidak dapat dimulai jika *Floodlight Controller* tidak ada saat jaringan simulasi aktif [21].

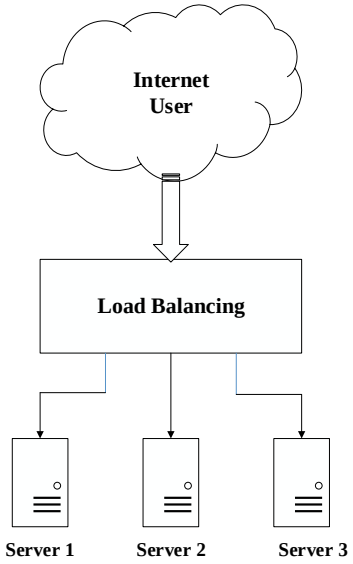
### 2.2.2 Web *Server*

*Server* web adalah perangkat lunak yang menawarkan layanan sebagai data. *Server web* berfungsi dengan menerima permintaan HTTP atau HTTPS dari klien, juga disebut sebagai *browser web* (*Chrome, Firefox*), dan kemudian menanggapi permintaan dengan mengirimkan halaman web ke klien. Data ini sering disimpan dalam format standar bahasa markup generik (SGML). Data kemudian akan ditampilkan dengan cara ini oleh browser sesuai dengan kemampuannya. [21].

### 2.2.3 *Load balancing*

*Load balancing* meningkatkan kinerja seluruh kluster dengan membagi beban di antara *node cluster*. Algoritma *load balancing* statis merumuskan strategi alokasi berdasarkan konfigurasi *server*. Algoritma *load balancing* dinamis memperoleh informasi kapasitas beban *server cluster* secara *real time*, dan secara dinamis mendistribusikan permintaan ke *node* dengan beban yang lebih ringan. Dari Sudut pandang kinerja *cluster* secara keseluruhan, efek peningkatan lebih baik daripada dampak *overhead* tambahan pada sistem. *Cluster server* adalah struktur yang efektif untuk mewujudkan jaringan yang sangat skalabel dan sangat tersedia layanan. Ketika beberapa *node* dalam *cluster* kelebihan beban atau gagal, penjadwal beban akan dikirim permintaan ke *node* dengan beban yang lebih ringan. Proses ini tidak memerlukan partisipasi operasi dan personel pemeliharaan untuk secara otomatis menyelesaikan penyesuaian beban [22].

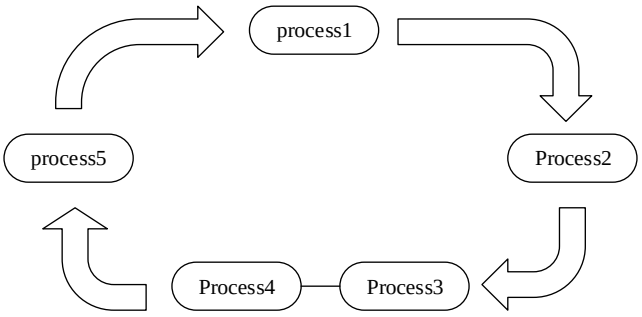
*Load balancing* merupakan salah satu metode untuk mendistribusikan beban *server* melalui beberapa *server*. Gambar 2.4 menunjukkan bawah sistem *Load balancing* membagi beban *server* secara merata di dua atau lebih jalur koneksi untuk memungkinkan *server* berfungsi sebaik mungkin. Layanan penyeimbangan beban memungkinkan akses ke sumber daya jaringan untuk didistribusikan di antara banyak *server* lain daripada terpusat, menghasilkan kinerja jaringan yang stabil secara keseluruhan. ketika jumlah pengguna di *server* melebihi apa yang dapat ditangani *server*, penyeimbangan beban biasanya dilakukan. Beban kerja setiap *server* berkurang melalui teknologi penyeimbangan beban, mencegah *server* yang kelebihan beban [23].




**Gambar 2.4** ***Server load balancing*** [24]

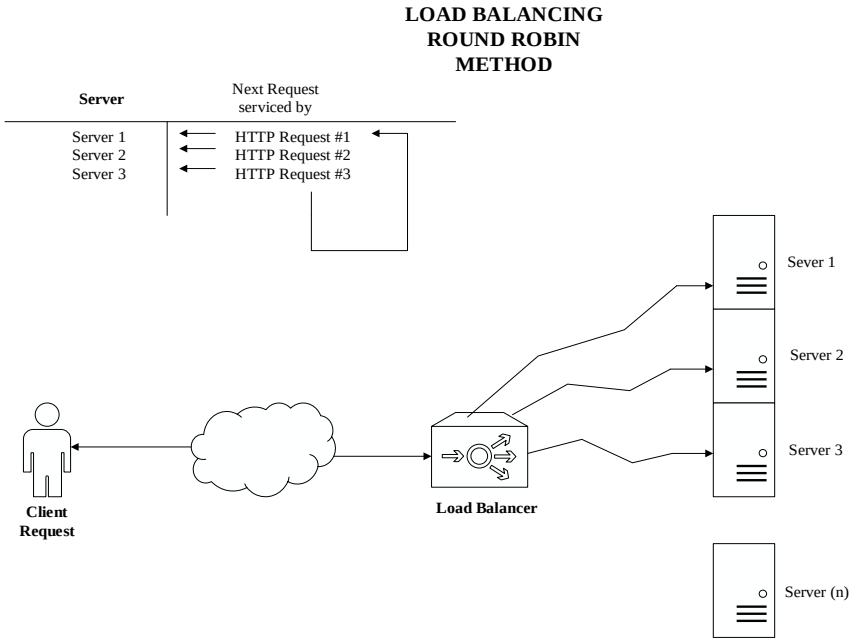
### 2.2.4 *Round robin*

*Round robin* adalah teknik untuk menjadwalkan operasi yang membagi setiap proses menjadi potongan waktu yang sama. Proses dengan periode pemrosesan yang lebih pendek akan selesai sebelum proses dengan waktu pemrosesan yang lebih lama. Dengan menggunakan kuantum yang sama di setiap operasi, teknik ini membagi proses. Berapa lama nilai kuantum proses berlangsung setelah ditentukan sebelumnya. Proses dalam metode ini menerima kuantum atau irisan waktu yang sama terlepas dari prioritasnya. Untuk mengakomodasi kebutuhan pembagian waktu, *round robin* dibuat. Semua proses dalam algoritma *round robin* ini berbagi waktu kuantum yang sama dengan proses pada gambar 2.5 yang menggambarkan algoritma *round robin* beraksi. [25].




**Gambar 2.5 Proses** ***Round robin***

Salah satu metode yang sering digunakan dalam *load balancing* adalah algoritma *round robin*. Gambar 2.6 memperlihatkan metode pada algoritma *round robin* yang bekerja secara berurutan dan bergiliran memindahkan beban dari satu *server* ke *server* lain di Traf IC. Ide dasar algoritma *round robin* adalah berbagi waktu. Untuk membatasi durasi pemrosesannya, setiap proses menerima waktu CPU yang dikenal sebagai waktu kuantum, yang biasanya 1-100 ms, algoritma ini melewati antrian satu per satu.

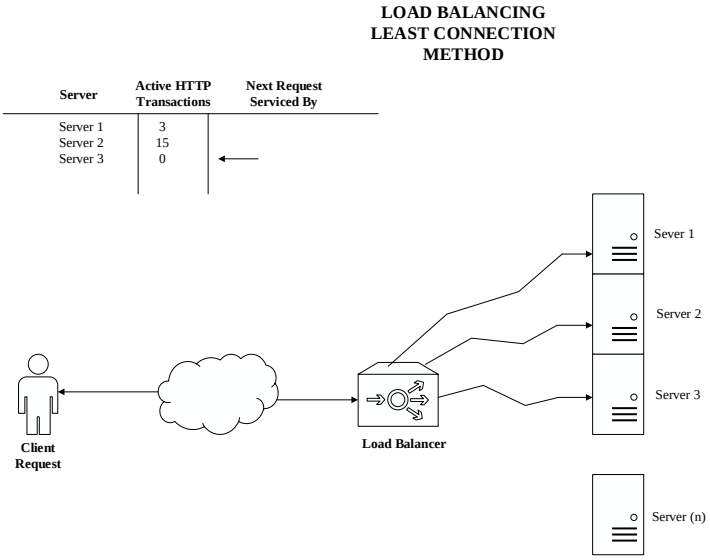



**Gambar 2.6 Proses Kerja Algoritma *Round robin*** [26].

Algoritma *round robin* beroperasi dengan membagi koneksi masuk secara merata di antara setiap grup *backend* di *server* yang sebenarnya. *Round-robin* mendistribusikan beban kerja di antara semua *node server* yang diperlukan dengan cara yang dikoordinasikan dengan distribusi beban setiap *server*. Ide dasar di balik algoritma ini adalah berbagi waktu, yang bertujuan untuk menyediakan proses antrian secara bergantian. Tidak ada batasan jumlah *server* aktif yang ditetapkan sebagai backend, oleh karena itu pendekatan *round robin* tidak mengizinkan pemindahan beban dinamis karena semuanya didefinisikan secara statis di awal. [26].

### 2.2.5 *Least connection*

Berdasarkan jumlah koneksi yang ditangani *server* saat ini, algoritma *Least connection* mengeksekusi algoritma penjadwalan beban. *Server* dengan jumlah koneksi terbesar akan menerima beban berikutnya, dan *server* dengan jumlah koneksi terbesar juga akan diturunkan ke *server* tingkat yang lebih rendah. Algoritma ini, yang secara dinamis menghitung koneksi aktif untuk setiap *server* yang sebenarnya. salah satu algoritma penjadwalan ketika ada banyak permintaan. Teknik penjadwalan ini bekerja dengan baik untuk distribusi yang mulus [27].




**Gambar 2.7 Proses *Least connection*** [28]

Gambar 2.7 secara jelas menggambarkan bahwa algoritma *least connection* memiliki kemampuan untuk membagi beban dengan proporsional berdasarkan jumlah koneksi yang mampu diatasi oleh setiap *server*. Penerapan algoritma ini pada kluster dengan konteks dinamis, di mana terjadi perubahan kondisi secara berkala, serta melakukan perubahan dalam alokasi sesi secara teratur, merupakan konsep yang sangat menarik dan berpotensi. Pendekatan ini menjanjikan kemampuan untuk mengoptimalkan pemanfaatan sumber daya kluster dengan lebih efisien, mengadaptasi diri terhadap perubahan permintaan, dan menjaga kinerja sistem pada tingkat optimal. Dengan demikian, melibatkan algoritma koneksi paling sedikit dalam lingkungan dan responsivitas keseluruhan sistem [29].