

## BAB II TINJAUAN PUSTAKA

### A. Penelitian Terkait

Penelitian yang digunakan sebagai referensi dan berkaitan dengan penelitian ini ditunjukkan pada Tabel 1 berikut.

Tabel 1 Penelitian Terkait

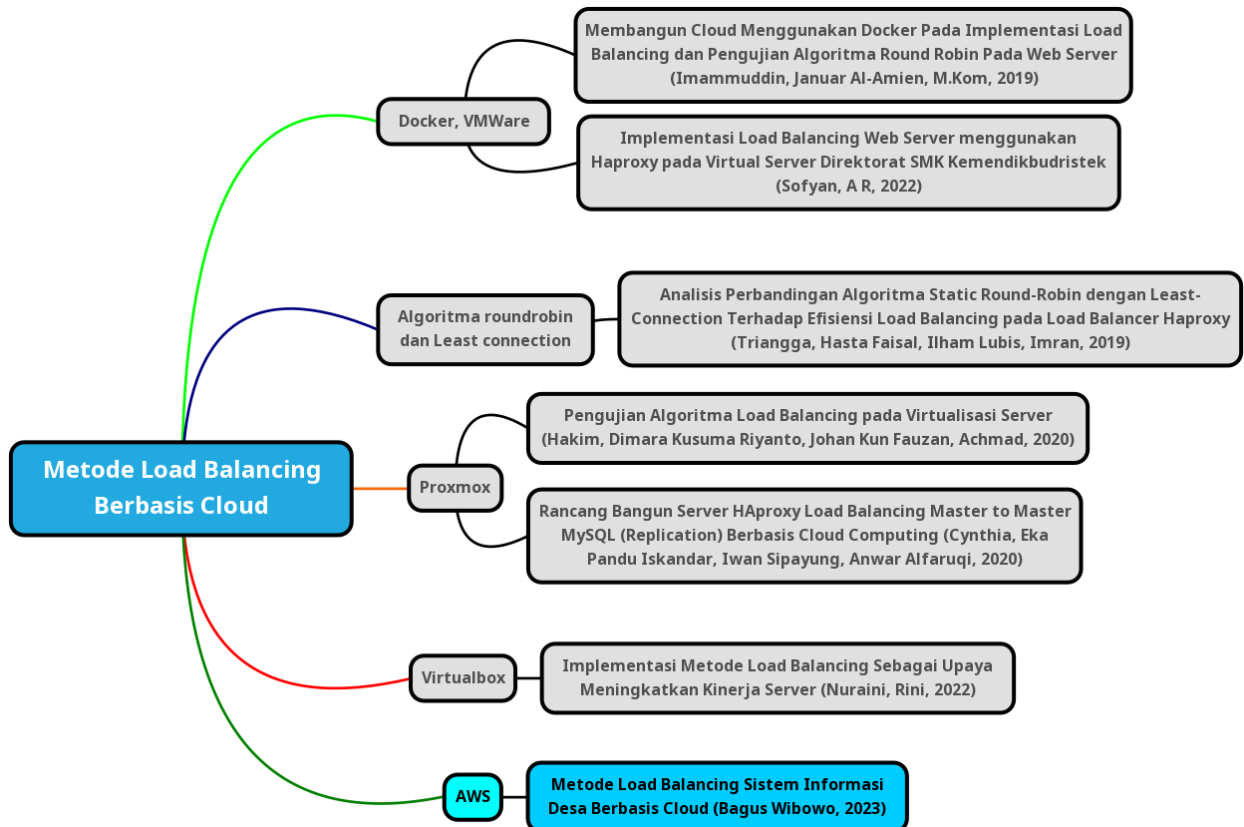
No	Judul (Penulis,tahun)	Keterangan
1	“Membangun <i>Cloud</i> Menggunakan Docker Pada Implementasi <i>Load Balancing</i> dan Pengujian Algoritma Round Robin Pada <i>Web Server</i> ” (Imammuddin, Januar Al-Amien, M.Kom, 2019)	Di penelitian ini, peneliti membangun <i>cloud</i> menggunakan docker dengan mengimplementasikan metode <i>load balancing</i> dan pengujian algoritma round robin pada <i>web server</i> . Peneliti juga membandingkan performa dari dua teknologi <i>cloud</i> yaitu <i>container</i> docker dan VMWare[8]. <b>Hasil</b> : Dari penggunaan Virtualisasi berbasis <i>container</i> pada <i>web server</i> di dapatkan hasil penggunaan memori pada <i>service</i> 1 hanya sebanyak 1.20Gb, dan jika menjalankan 2 <i>service</i> sekaligus penggunaan memori hanya sebanyak 1.43 GB. Pengambilan <i>resource</i> ke <i>kernel Host OS</i> menjadikan virtualisasi berbasis <i>container</i> sangat ringan dan hanya mengambil <i>service-service</i> yang di perlukan sehingga <i>resource</i> dari <i>Host OS</i> tidak terlalu terbebani, dan <i>service</i> satu dengan <i>service</i> yang lain terisolasi sehingga <i>service</i> tidak akan terganggu. Sedangkan untuk VMware dengan pengujian 3 <i>Virtual server</i> Ubuntu mengkonsumsi jumlah memori berjumlah 1.73Gb
2	“Analisis Perbandingan Algoritma Static Round Robin dengan Least Connection Terhadap Efisiensi <i>Load Balancing</i> pada <i>Load Balancer</i> HAProxy”	Di penelitian ini, peneliti melakukan analisis perbandingan static round robin dengan least connection terhadap efisiensi metode <i>load balancing</i> pada HAProxy[9]. <b>Hasil</b> : HAProxy bertindak sebagai <i>proxy</i> terbalik yang diakses oleh klien sementara <i>server backend</i> menangani permintaan <i>HTTP</i> . Eksperimen tersebut melibatkan 20

No	Judul (Penulis,tahun)	Keterangan
	(Triangga, Hasta Faisal, Ilham Lubis, Imran, 2019)	<p><i>PC Client</i> yang digunakan untuk melakukan permintaan <i>HTTP</i> secara bersamaan, menggunakan algoritma Static round robin dan Least connection pada <i>load balancer</i> HAProxy secara bergantian. Saat menggunakan algoritma Static round robin, didapatkan hasil persentase rata-rata penggunaan <i>CPU</i> berturut-turut selama 1 menit; 5 menit; dan 15 menit adalah; 0,1%; 0,25%; dan 1,15% dengan rata-rata <i>throughput</i> yang dihasilkan adalah 14,74 kbps. Rata-rata total <i>delay</i> yang dihasilkan adalah 64,3 kbps. Rata-rata total <i>delay</i> dan <i>jitter</i> masing-masing adalah 181,3 ms dan 11,1 ms. Sedangkan untuk algoritma Least connection diperoleh persentase rata-rata berturut-turut selama 1 menit; 5 menit; dan 15 menit adalah 0,1%; 0,3%; dan 1,25% dengan rata-rata <i>throughput</i> yang dihasilkan adalah 14,66 kbps. Rata-rata total <i>delay</i> dan <i>jitter</i> masing-masing adalah 350,3 ms dan 24,5 ms. Ini berarti algoritma Static round robin lebih efisien daripada algoritma least connection karena dapat menghasilkan <i>throughput</i> yang lebih besar dengan beban <i>CPU</i> yang lebih sedikit dan penundaan total yang lebih sedikit.</p>
3	<p>“Pengujian Algoritma <i>Load Balancing</i> pada Virtualisasi <i>Server</i>” (Hakim, Dimara Kusuma Riyanto, Johan Kun Fauzan, Achmad, 2020)</p>	<p>Pada penelitian ini, peneliti melakukan pengujian algoritma pada metode <i>load balancing</i> di virtualisasi <i>server</i>. Algoritma yang dibandingkan yaitu weighted round robin, round robin dan least connection[1]. <b>Hasil</b> : Hasil dari uji kecepatan akses tersebut di uji lagi perbedaan rata-rata menggunakan uji ANOVA, yang mana memungkinkan peneliti untuk menguji hipotesis perbandingan rata-rata lebih dari dua kelompok dari algoritma round robin, weighted round robin dan least connection dengan hasil, bahwa pada saat 25 dan 55 <i>user</i></p>

No	Judul (Penulis,tahun)	Keterangan
		<p>algoritma round robin lebih baik dari pada weighted round robin dan least connection, namun pada saat 75, 100, 125, 250 dan 1000 <i>user</i> algoritma weighted round robin lebih baik dan optimal dari pada algoritma round robin dan least connection pada saat akses ribuan data dari <i>web server</i> dan algoritma least connection tidak memberikan performa yang baik terbukti dari ketujuh pengujian dan simulasi yang berbeda tetap tidak memberikan hasil rata-rata yang kurang dari algoritma round robin dan weighted round robin. Dari ketujuh simulasi yang telah dilakukan. Maka penggunaan algoritma weighted round robin lebih baik dari pada round robin dan least connection</p>
4	<p>“Rancang Bangun <i>Server HAProxy Load Balancing Master to Master MySQL (Replication)</i> Berbasis <i>Cloud Computing</i>” (Cynthia, Eka Pandu Iskandar, Iwan Sipayung, Anwar Alfaruqi, 2020)</p>	<p>Pada penelitian ini, peneliti membuat rancang bangun <i>server HAProxy</i> dengan metode <i>load balancing master to master mysql (replication)</i> berbasis <i>cloud computing</i>[10]. <b>Hasil :</b> Dari hasil pengujian yang dilakukan pada penelitian ini, <i>server HAProxy</i> dinilai mampu menangani permasalahan dibandingkan dengan <i>server</i> tunggal. Replikasi yang diuji juga dinilai dapat mejadi solusi untuk menjembatani perubahan data yang ada pada <i>server HAProxy</i>.</p>
5	<p>“Implementasi <i>Load Balancing Web Server</i> menggunakan <i>HAProxy</i> pada Virtual Server Direktorat SMK Kemendikbudristek” (Ahmad Riyan Sofyan, Susanna Dwi Yulianti Kusuma, 2022)</p>	<p>Pada penelitian ini, peneliti melakukan implementasi <i>load balancing web server</i> menggunakan <i>HAProxy</i> dengan algoritma round robin pada <i>virtual server</i> Direktorat SMK kemendikbudristek. Peneliti menggunakan metode <i>NLDC (Network Development Life Cycle)</i> pada penelitiannya[2]. <b>Hasil :</b> Hasil dari penelitian ini menghasilkan perhitungan dengan menggunakan <i>HAProxy</i> metode algoritma round robin dimana beban kerja <i>server</i> dapat berjalan seimbang dengan</p>

No	Judul (Penulis,tahun)	Keterangan
		cara memberikan bobot ke masing-masing <i>server</i> atau <i>node cluster</i> sehingga dapat memenuhi permintaan atau <i>request</i> dari pengguna
6	“Implementasi Metode <i>Load Balancing</i> Sebagai Upaya Meningkatkan Kinerja <i>Server</i> ” (Nuraini, Rini, 2022)	Pada penelitian ini, peneliti mengimplementasi metode <i>load balancing</i> sebagai upaya meningkatkan kinerja <i>server</i> [11]. <b>Hasil</b> : Dalam pengujian yang telah dilakukan, skalabilitas sistem menjadi meningkat. Ini dibuktikan ketika sistem dengan <i>load balancing</i> diberikannya 10000 koneksi, hasil pengujian memiliki nilai rata-rata <i>response time</i> sebesar 44,42 ms. Sedangkan untuk sistem tanpa <i>load balancing</i> , hasil pengujian memiliki nilai rata-rata nilai <i>response time</i> sebesar 185,88 ms. Dari hasil pengujian terlihat bahwa nilai rata-rata <i>response time</i> dari sistem <i>server</i> dengan <i>load balancing</i> lebih kecil dibandingkan tanpa <i>load balancing</i> , maka kinerja layanan dari sistem dapat terus ditingkatkan dengan diterapkannya <i>load balancing</i> .

Ada beberapa hal yang membedakan penelitian ini dengan penelitian sebelumnya yaitu: pada penelitian ini, peneliti menggunakan teknologi *cloud* dengan platform Amazon Web Services (AWS) sedangkan di penelitian sebelumnya menggunakan teknologi *cloud* seperti container docker, VMWare, dengan berbasis *localhost*. Pada penelitian ini juga terdapat beberapa metode algoritma pada HAProxy yang berbeda dari penelitian sebelumnya yaitu algoritma source dan URI. Begitupun dengan *software* untuk menguji kinerja *web server*, di penelitian ini *software* yang digunakan adalah Apache JMeter versi 5.5. Untuk lebih jelasnya, disajikan gambaran pada diagram *mind map* dibawah ini.



Gambar 1 *Mind map* penelitian sebelumnya

## B. Landasan Teori

### 1. *Cloud Computing*

*Cloud Computing* adalah teknologi komputasi yang menggunakan layanan internet dalam mengakses *resources*. *Infrastructure as a Service (IaaS)* merupakan salah satu layanan *cloud computing* yang menyewakan infrastruktur *IT* berupa *storage*, *networks* dan sumber daya komputasi lainnya[6]. Sumber daya komputasi dari *cloud computing* tersebar dan dapat diakses berdasarkan kebutuhan dari perangkat apapun dan dimanapun terhubung. Mell dan Grace[12] menjelaskan *cloud computing* terdapat 5 karakter penting yaitu diantaranya:

- a. *On-Demand* dimana pengguna dapat memesan dan mengelola layanan tanpa interaksi manusia dengan penyedia layanan.
- b. *Broad Network* merupakan kemampuan yang tersedia melalui jaringan dan di akses melalui mekanisme standar, yang mengenalkan pengguna sebagai platform.
- c. *Resource Pooling* merupakan memadukan antara sumber daya komputasi yang dimiliki oleh pengguna untuk melayani beberapa konsumen.
- d. *Rapid Elasticity* adalah kemampuan yang dapat dengan cepat dan elastis ditetapkan.

- e. *Measured Service* adalah *cloud computing* yang secara otomatis mengawasi dan mengoptimalkan penggunaan sumber daya dengan memanfaatkan kemampuan pengukuran pada beberapa tingkat yang sesuai.

*Cloud computing* memiliki karakteristik yang salah satunya adalah *scalable*, yaitu dapat melakukan penyesuaian kapasitas maupun *traffic* yang cukup besar. *Cloud computing* menyediakan suatu layanan infrastruktur yang menjadi data pusat secara terdistribusi serta bersifat masif yang terkoneksi dengan *IP* jaringan. *Cloud computing* terbagi menjadi tipe layanan seperti berikut[12]:

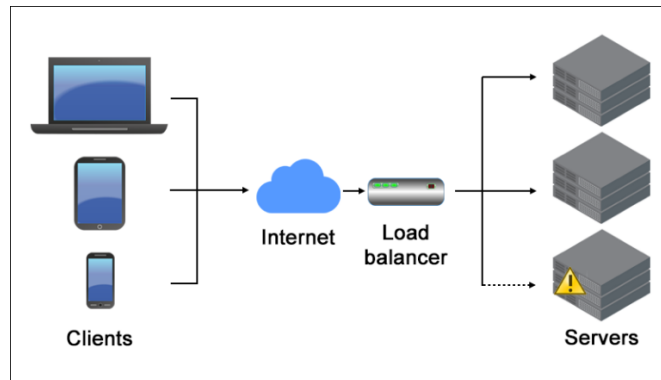
- a. *IaaS (Infrastructure as a Service)* yaitu layanan yang sistem operasi dijadikan sebagai virtual *machine*, sehingga seorang *developer* dapat melakukan komputasi hingga pada level sistem operasi,
- b. *PaaS (Platform as a Services)* yaitu memberikan layanan pada level aplikasi seperti penyediaan platform *database*, penyediaan *web server* hingga aplikasi lainnya. Aplikasi memudahkan pengguna dalam melakukan konfigurasi terhadap kebutuhan yang diinginkan, karena aplikasi atau platform yang diberikan memiliki panel-panel yang mudah untuk dapat melakukan konfigurasi sesuai dengan kebutuhan,
- c. *SaaS (Software as a Service)* yaitu layanan yang menyediakan suatu bentuk aplikasi yang sudah jadi kepada para *developer*. Contoh dari *software as a service* adalah seperti Gmail, Google Docs dan Drop Box.

## 2. *Load Balancing*

*Load balancing* adalah suatu teknik penyeimbangan yang digunakan pada *cloud computing* untuk mengatasi beban dalam jaringan dengan skala yang besar dan juga *traffic* daya yang tinggi. Ketika *user* mengakses suatu *web*, maka pada saat itu *server* mengalami beban *traffic* karena permintaan oleh *user*. Semakin banyak *user* yang melakukan *request* maka semakin tinggi *traffic* yang diberikan. *Load balancing* dapat menyeimbangkan beban tersebut sehingga tidak terjadi *overload*[13]. *Load balancing* dibutuhkan supaya tidak terjadi kegagalan yang dapat mengakibatkan tidak tersedianya data. Manfaat dari *load balancing* diantaranya adalah menjamin reliabilitas layanan, sehingga mampu melayani *user* dengan baik dan menjaga ketersediaan dari *web* karena memiliki *server* lebih dari satu. Sehingga jika *server* satu mati maka akan dilakukan penambahan *server* untuk meningkatkan skalabilitasnya. *Load balancing* mendistribusikan *traffic* yang tinggi pada dua atau lebih jalur secara seimbang.

*Load balancing* dapat dilakukan di beberapa lapisan dalam arsitektur aplikasi, seperti lapisan jaringan, *transport*, aplikasi, atau *database*. Di lapisan jaringan, *load balancing* dilakukan dengan memperkenalkan perangkat khusus yang dikenal sebagai *load balancer*, yang berfungsi sebagai titik masuk tunggal

ke jaringan untuk semua permintaan dari klien. Perangkat ini akan mendistribusikan permintaan ke sumber daya di belakangnya dengan cara yang merata. Di lapisan aplikasi, *load balancing* dilakukan dengan menggunakan teknik algoritma seperti round robin, least connections, atau source, yang memungkinkan permintaan dari klien di distribusikan ke beberapa *server* secara merata. Secara keseluruhan, *load balancing* sangat penting untuk memastikan ketersediaan, keandalan, dan kinerja sistem, terutama di lingkungan yang sangat sibuk[14]. Dengan memperkenalkan teknik *load balancing*, kita dapat meningkatkan efisiensi penggunaan sumber daya komputasi dan memastikan bahwa permintaan pengguna dapat diakomodasi dengan baik.



Gambar 2 Ilustrasi cara kerja *load balancing*

(sumber: Cahya Kurniawan[13])

### 3. HAProxy

High Availability Proxy adalah kepanjangan dari HAProxy sebuah perangkat lunak *open source* dibawah GPLv2 license. HAProxy digunakan untuk membagi beban *request* atau *load balancer TCP/HTTP* dan solusi *proxy* yang dapat dijalankan di sistem operasi Linux, Solaris, dan FreeBSD[4]. HAProxy ditulis dalam bahasa C oleh Willy Tarreau yang didukung oleh *SSL*, *keep-alive custom log format*, dan *header writing*. HAProxy dapat menangani banyak jenis protokol lalu lintas, termasuk *HTTP*, *HTTPS*, *TCP*, dan *SSL/TLS*. HAProxy dirancang untuk mengoptimalkan performa aplikasi *web* dengan menggunakan teknik seperti *SSL/TLS offloading*, *caching*, dan kompresi. Dalam pengaturan distribusi beban kerja, HAProxy membagi lalu lintas yang diterima di antara beberapa *server*, sehingga mencegah satu *server* menjadi *over-utilized* dan menjamin ketersediaan aplikasi. HAProxy tersedia sebagai solusi *open-source* yang dapat diunduh dan digunakan secara gratis. HAProxy dapat diinstal pada sistem operasi berbasis Linux dan BSD, dan banyak *vendor cloud* juga menyediakan HAProxy sebagai solusi *load balancer* terkelola.

Cara kerja HAProxy adalah sebagai berikut:

- a. HAProxy menerima permintaan dari klien, baik itu permintaan *HTTP* atau permintaan lainnya. Permintaan ini datang ke HAProxy melalui *IP* dan *port* tertentu yang telah ditetapkan sebelumnya.
- b. Setelah menerima permintaan dari klien, HAProxy meneruskannya ke *server backend* yang sesuai. *Server backend* dapat dipilih menggunakan algoritma *load balancing* yang telah ditentukan sebelumnya. Misalnya, HAProxy dapat memilih *server backend* dengan beban kerja yang paling rendah menggunakan algoritma *least connections*.
- c. Jika *server backend* yang dipilih tidak tersedia atau mengalami gangguan, HAProxy akan meneruskannya ke *server backend* lain yang tersedia. Hal ini dilakukan untuk memastikan ketersediaan dan keandalan sistem.
- d. Setelah menerima *respons* dari *server backend*, HAProxy mengirimkannya kembali ke klien. *Respons* ini dapat diubah atau dimodifikasi oleh HAProxy menggunakan fitur-fitur seperti *SSL termination*, *HTTP accelerator*, atau *caching*.
- e. HAProxy juga dapat melakukan *logging* dan *monitoring* terhadap aktivitasnya, seperti jumlah koneksi, permintaan *HTTP*, dan kesalahan. Hal ini berguna untuk analisis dan pemecahan masalah di kemudian hari.

Dalam beberapa kasus, HAProxy juga dapat berfungsi sebagai *reverse proxy*. Dalam hal ini, HAProxy akan memproses permintaan dari klien dan meneruskannya ke *server backend* yang sesuai. *Server backend* tidak terlihat oleh klien dan hanya dapat diakses melalui HAProxy. Secara umum, cara kerja HAProxy sangat bergantung pada konfigurasi yang telah ditentukan sebelumnya. Konfigurasi ini dapat disesuaikan sesuai dengan kebutuhan dan lingkungan aplikasi yang digunakan. Dalam penggunaannya, HAProxy dapat membantu meningkatkan kinerja dan ketersediaan aplikasi *web* secara signifikan. Alasan kenapa pada penelitian ini menggunakan HAProxy sebagai *load balancer* yaitu karena ada beberapa alasan, diantaranya adalah: yang pertama karena HAProxy gratis, alasan selanjutnya yaitu karena HAProxy mudah dikonfigurasi dan fleksibel, konfigurasi HAProxy juga dapat diubah secara dinamis tanpa perlu memulai ulang atau menghentikan HAProxy.

#### 4. Amazon Web Services (AWS)

Amazon Web Services (AWS) adalah anak perusahaan dari raksasa *e-commerce* dunia yaitu Amazon. Amazon sebelumnya lebih terkenal dengan toko buku *online*-nya, kemudian pada tahun 2005 Amazon mengembangkan dirinya menjadi Amazon Web Services (AWS) yang menyediakan layanan *cloud computing*[7]. AWS adalah platform *cloud* yang komprehensif dan dapat



digunakan secara luas di dunia, memberikan suatu penawaran lebih dari 175 layanan unggulan yang lengkap dari pusat data secara global. AWS memiliki lebih banyak fitur dalam layanannya, dibandingkan dengan penyedia *cloud* lainnya, mulai dari teknologi infrastruktur perhitungan, penyimpanan sampai ke *database* hingga teknologi yang berkembang seperti *machine learning* dan *artificial intelligence*[14].

AWS dirancang untuk memberikan layanan yang mudah digunakan, cepat, dan terukur, dengan biaya yang rendah dan dapat disesuaikan. AWS juga menawarkan berbagai fitur keamanan yang membantu pengguna menjaga privasi dan keamanan data mereka. Beberapa layanan utama yang disediakan oleh AWS antara lain:

- a. *Elastic Compute Cloud (EC2)*: Layanan ini menyediakan kapasitas komputasi yang dapat disesuaikan untuk menjalankan aplikasi di dalam lingkungan *cloud*. Pengguna dapat memilih jenis *server*, sistem operasi, dan kapasitas penyimpanan yang diinginkan. Layanan *EC2* memungkinkan pengguna untuk menyewa *server* virtual dengan berbagai spesifikasi dan konfigurasi yang dapat diubah-ubah sesuai kebutuhan. Layanan ini memberikan fleksibilitas yang tinggi untuk mengelola beban kerja dan infrastruktur *server*.
- b. *Relational Database Service (RDS)*: Layanan ini menyediakan *database* relasional yang dapat disesuaikan dan *scalable*. *RDS* dapat digunakan untuk menyimpan data transaksional, seperti data pelanggan atau data keuangan. *RDS* menyediakan layanan *database* yang dikelola secara penuh, seperti MariaDB, PostgreSQL, dan Oracle. Layanan ini memudahkan pengguna untuk mengelola *database* dengan mudah dan dapat diandalkan.
- c. *Virtual Private Cloud (VPC)*: Layanan ini menyediakan jaringan virtual yang terisolasi dan dapat disesuaikan. Pengguna dapat mengatur jaringan yang hanya dapat diakses oleh karyawan atau pengguna yang diizinkan. *VPC* adalah layanan yang memungkinkan pengguna untuk membuat jaringan virtual yang aman dan dapat disesuaikan sesuai kebutuhan. Layanan ini memungkinkan pengguna untuk menghubungkan *server EC2* dan layanan AWS lainnya dalam jaringan yang aman.

AWS terus berkembang dan menambahkan layanan baru secara berkala. AWS juga memiliki ekosistem yang besar dari *partner* dan pengembang yang dapat membantu pengguna dalam membangun dan mengelola aplikasi mereka di dalam lingkungan *cloud*. AWS memiliki beberapa kelebihan dibandingkan dengan layanan *cloud computing* lainnya, di antaranya:

- a. Skala global: AWS memiliki jaringan pusat data yang sangat luas dan tersebar di seluruh dunia, sehingga memungkinkan pelanggan untuk

memilih pusat data yang paling dekat dengan lokasi mereka, sehingga meningkatkan kinerja aplikasi mereka.

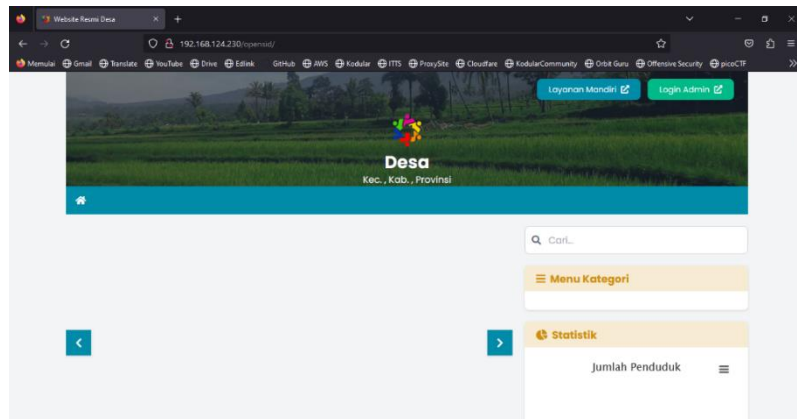
- b. **Fleksibilitas:** AWS menawarkan berbagai macam layanan *cloud computing*, termasuk infrastruktur, platform, dan perangkat lunak sebagai layanan. Pelanggan dapat memilih layanan yang paling sesuai dengan kebutuhan bisnis mereka dan dapat dengan mudah menyesuaikan layanan tersebut seiring dengan perkembangan bisnis mereka.
- c. **Ketersediaan:** AWS menawarkan jaminan ketersediaan layanan hingga 99,99%, yang berarti bahwa layanan akan tersedia hampir sepanjang waktu tanpa adanya gangguan yang signifikan.
- d. **Keamanan:** AWS memiliki lapisan keamanan yang sangat kuat dan terus meningkatkan keamanan sistem dan data. AWS juga menyediakan berbagai layanan keamanan tambahan, seperti *AWS Shield*, *AWS WAF*, dan *AWS GuardDuty*.
- e. **Harga yang kompetitif:** AWS menawarkan harga yang sangat kompetitif untuk layanan *cloud computing*. Pelanggan dapat mengontrol biaya dengan memilih layanan yang paling sesuai dengan kebutuhan mereka dan hanya membayar untuk layanan yang digunakan.
- f. **Integrasi dengan teknologi terkini:** AWS mendukung berbagai teknologi terkini, seperti *Internet of Things (IoT)*, *machine learning*, dan *big data*. Ini memungkinkan pelanggan untuk mengembangkan aplikasi yang sangat inovatif dan berkinerja tinggi.
- g. **Kemudahan penggunaan:** AWS menyediakan antarmuka pengguna yang mudah digunakan, sehingga pelanggan dapat dengan mudah mengelola dan memonitor aplikasi mereka di AWS.

Kelebihan-kelebihan tersebut menjadikan alasan peneliti menggunakan AWS sebagai penyedia layanan *cloud computing* yang akan digunakan.

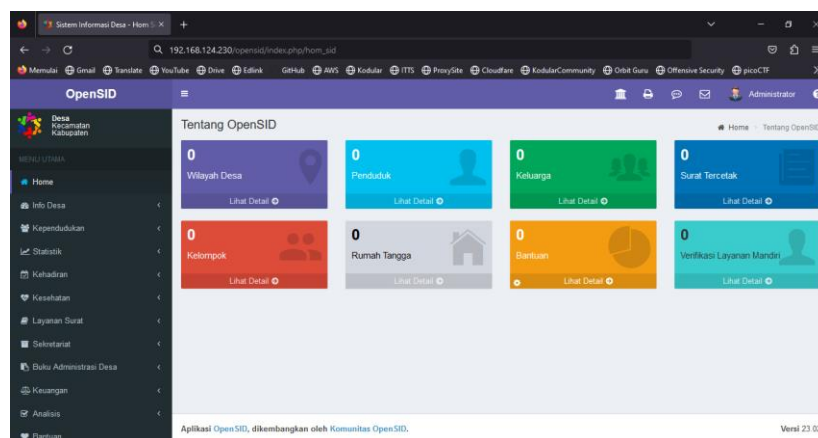
## 5. OpenSID

OpenSID adalah Sistem Informasi Desa (*SID*) yang dibuat secara terbuka dan dapat dikembangkan bersama-sama oleh komunitas peduli *SID*. Beberapa desa sudah menerapkan OpenSID[15]. Selain untuk menyimpan data penduduk, OpenSID juga bisa digunakan untuk *web* desa sebagai media promosi dan penyebaran informasi desa. Selain itu, OpenSID juga dapat diintegrasikan dengan aplikasi lain seperti *sms gateway*, *monitoring* detak jantung warga, dan penilaian kepuasan pengguna[15]. Ada tiga tujuan pengembangan OpenSID, yaitu: (1) Memudahkan pengguna untuk mendapatkan *SID* secara bebas, tanpa proses birokrasi; (2) Memudahkan pengguna menyerap rilis *SID* terbaru; dan (3) Memungkinkan pegiat *SID* untuk membuat kontribusi langsung pada kode sumber aplikasi *SID*. OpenSID

diharapkan dapat membantu pemerintah desa dalam beberapa hal berikut: kantor desa lebih efisien dan efektif, pemerintah desa lebih transparan dan akuntabel, layanan publik lebih baik, dan warga mendapat akses lebih baik pada informasi desa. OpenSID sudah digunakan sebanyak 6.092 desa dengan rincian 3.513 desa sudah *online*, dan 4.175 masih *offline*[15]. Berdasarkan info tersebut, total penggunaan *SID* (*SIDeKa* dan *OpenSID*) sebanyak 12.628 dari 74.957 desa seluruh Indonesia, jadi masih ada sekitar 83,15% desa yang belum mempunyai *SID*.



Gambar 3 Halaman depan OpenSID



Gambar 4 Halaman admin OpenSID

OpenSID adalah sebuah aplikasi Sistem Informasi Desa yang bersifat *open source* atau sumber terbuka, yang memiliki beberapa keunggulan antara lain:

- a. **Gratis:** OpenSID dapat diunduh, digunakan, dan dimodifikasi secara gratis tanpa perlu membayar biaya lisensi.
- b. **Sumber terbuka:** Karena bersifat *open source*, OpenSID memungkinkan pengembang lokal maupun internasional untuk memodifikasi dan mengembangkan sistem sesuai kebutuhan.
- c. **Mudah digunakan:** OpenSID dirancang dengan antarmuka yang sederhana dan mudah dipahami oleh pengguna awam sehingga

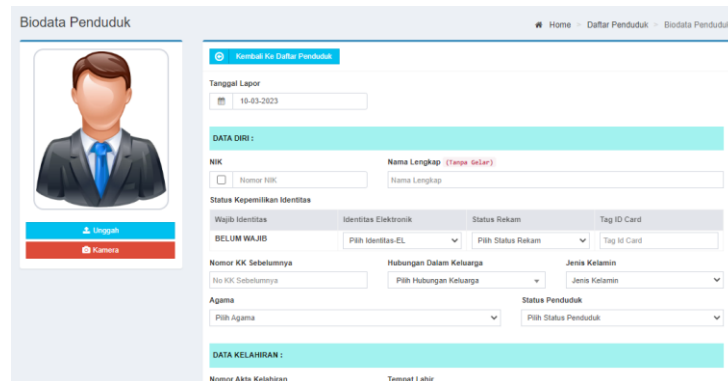
memudahkan pihak desa dalam mengelola informasi administrasi desa.

- d. Modular: OpenSID memiliki banyak modul yang bisa digunakan sesuai kebutuhan desa, seperti modul administrasi, modul keuangan, modul pengelolaan data penduduk, modul informasi publik, dan lain-lain.
- e. Terintegrasi: OpenSID dapat diintegrasikan dengan aplikasi lain, seperti aplikasi *e-government*, sehingga memudahkan desa dalam memberikan pelayanan publik kepada masyarakat.
- f. Fleksibel: OpenSID dapat diinstal dan digunakan pada berbagai sistem operasi dan *web server*, sehingga dapat diakses dari berbagai perangkat.
- g. Komunitas yang aktif: OpenSID didukung oleh komunitas yang aktif dalam mengembangkan dan memperbaiki aplikasi ini, sehingga aplikasi ini terus berkembang dan ditingkatkan fungsinya.

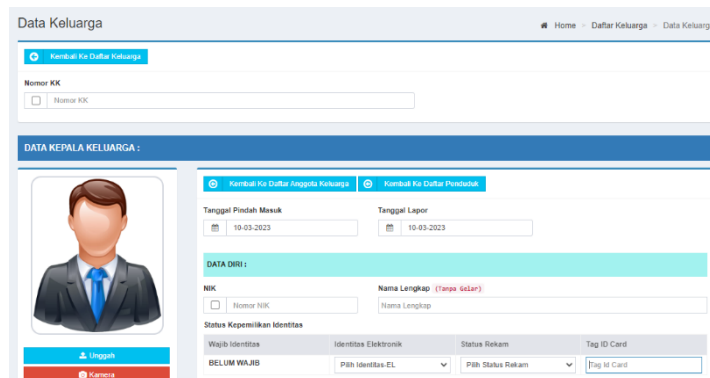
Beberapa fitur yang disediakan oleh OpenSID antara lain:

- a. Manajemen data penduduk: OpenSID dapat digunakan untuk mengelola data penduduk desa seperti nama, alamat, tanggal lahir, jenis kelamin, dan status perkawinan.
- b. Manajemen data kependudukan: Selain data penduduk, OpenSID juga dapat digunakan untuk mengelola data kependudukan seperti kartu keluarga, akta kelahiran, akta kematian, dan lain-lain.
- c. Manajemen data keuangan: OpenSID dapat digunakan untuk mengelola data keuangan desa seperti penerimaan dan pengeluaran, pembayaran gaji, dan lain-lain.
- d. Manajemen data proyek: OpenSID dapat digunakan untuk mengelola data proyek desa seperti pembangunan jalan, pembangunan irigasi, dan lain-lain.
- e. Pencatatan surat-menyurat: OpenSID dapat digunakan untuk mencatat surat-menyurat desa seperti surat keterangan domisili, surat keterangan kelahiran, surat keterangan kematian, dan lain-lain.
- f. Pembuatan *website* desa: OpenSID dapat digunakan untuk membuat *website* desa yang dapat diakses oleh masyarakat untuk mendapatkan informasi tentang desa dan kegiatan-kegiatan yang sedang dilaksanakan.

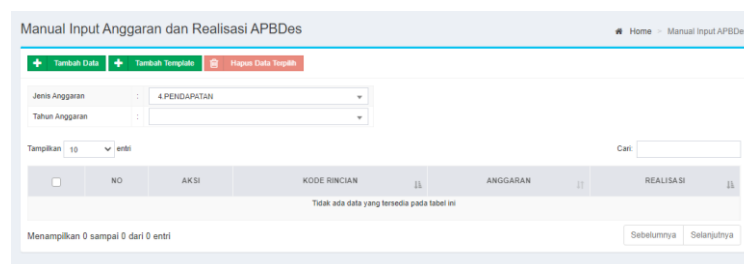
- g. Keterbukaan informasi publik: OpenSID dapat digunakan untuk mempublikasikan informasi tentang kegiatan-kegiatan desa dan anggaran desa sehingga masyarakat dapat mengetahui secara transparan penggunaan anggaran desa.



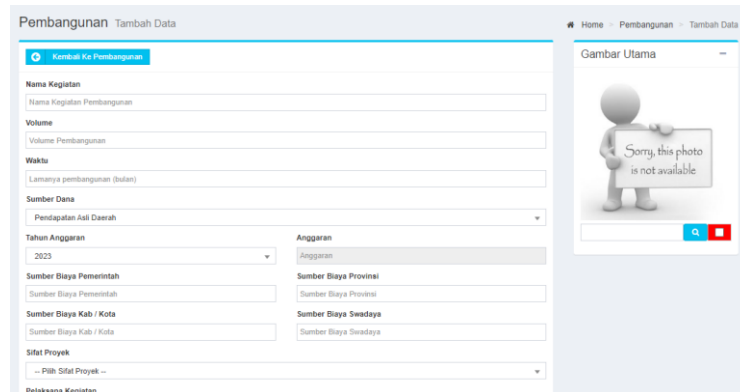
Gambar 5 Manajemen data penduduk di OpenSID



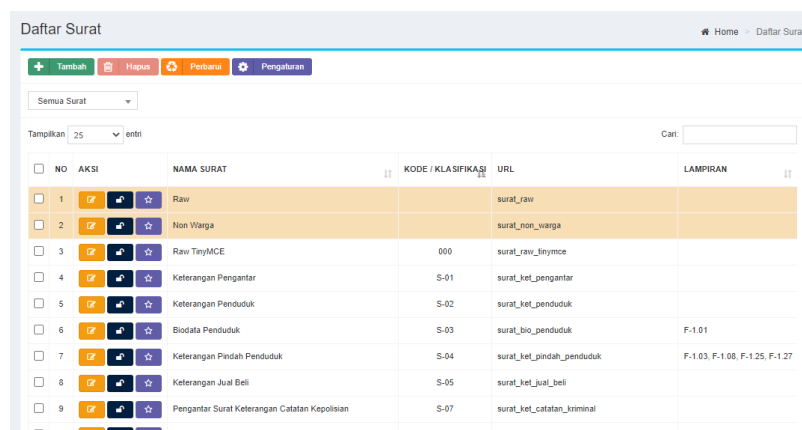
Gambar 6 Manajemen data kependudukan di OpenSID



Gambar 7 Manajemen data keuangan di OpenSID



Gambar 8 Manajemen data pembangunan di OpenSID



Gambar 9 Layanan surat di OpenSID

Alasan kenapa menggunakan OpenSID di penelitian ini yaitu karena selain gratis, OpenSID juga memiliki fitur yang cukup lengkap dan mudah diterapkan sebagai *website* sistem informasi desa.

## 6. Ubuntu Server

Ubuntu server adalah distribusi sistem operasi Linux yang dikembangkan oleh Canonical Ltd. dan didasarkan pada distribusi Debian. Ini adalah sistem operasi *open source* yang dirancang untuk digunakan sebagai platform *server* untuk berbagai keperluan, termasuk *hosting web*, penyimpanan data, pengembangan perangkat lunak, dan banyak lagi. Ubuntu server adalah "sistem operasi Linux yang kuat dan aman yang dikembangkan khusus untuk digunakan sebagai platform *server*[16]."

Ubuntu server juga memiliki fitur-fitur yang membuatnya populer di kalangan pengguna Linux, seperti dukungan untuk lingkungan pengembangan, kemampuan *clustering*, dan dukungan untuk beberapa arsitektur *hardware*. Ubuntu server juga menyediakan *tools* dan layanan manajemen sistem yang memungkinkan pengguna untuk mengelola sistem dengan mudah. Ubuntu server dapat diunduh secara gratis dari situs *web* resmi Ubuntu, dan pengguna dapat mengakses forum dukungan dan sumber daya lainnya untuk membantu

mereka mengatasi masalah yang mungkin mereka temukan saat menggunakan sistem operasi ini. Alasan kenapa memilih sistem operasi ini, karena memiliki beberapa keunggulan dibandingkan dengan sistem operasi lainnya yaitu mudah digunakan, Ubuntu server dirancang agar mudah digunakan dan diinstal serta dikonfigurasi, bahkan untuk pemula. Ini dilengkapi dengan antarmuka berbasis *web* yang sederhana untuk mengelola *server* dan layanannya. Selain mudah digunakan, Ubuntu server juga merupakan sistem operasi *open source*, yang berarti bahwa itu gratis untuk diunduh dan digunakan, dan kode sumbernya tersedia untuk siapa saja untuk dilihat dan dimodifikasi.

## 7. Apache

Apache merupakan sebuah *web server* berbasis UNIX yang dapat digunakan secara bebas. Apache mendukung berbagai macam fitur dan banyak diimplementasikan sebagai modul yang dapat diintegrasikan dengan aplikasi lainnya untuk meningkatkan fungsionalitas inti aplikasi tersebut. Apache Web Server merupakan *unix-based web server*, Apache awalnya dikembangkan berbasis kode pada NCSA HTTPD 1.3 yang kemudian di program ulang menjadi sebuah *web server* yang paling banyak digunakan saat ini[17]. Apache kini menjadi *web server* yang paling populer dan banyak digunakan lebih dari 42% dari berbagai *domain website* yang ada di *internet*[17]. Apache memiliki fitur yang sangat lengkap mulai dari performa yang tinggi, fungsionalitas, efisiensi, serta kecepatan. Alasan kenapa menggunakan Apache dalam penelitian ini adalah karena Apache kompatibel dengan banyak bahasa pemrograman seperti PHP, Python, dan Perl. Apache juga mudah ditemukan dan diunduh karena bersifat *open source*.

## 8. PHP

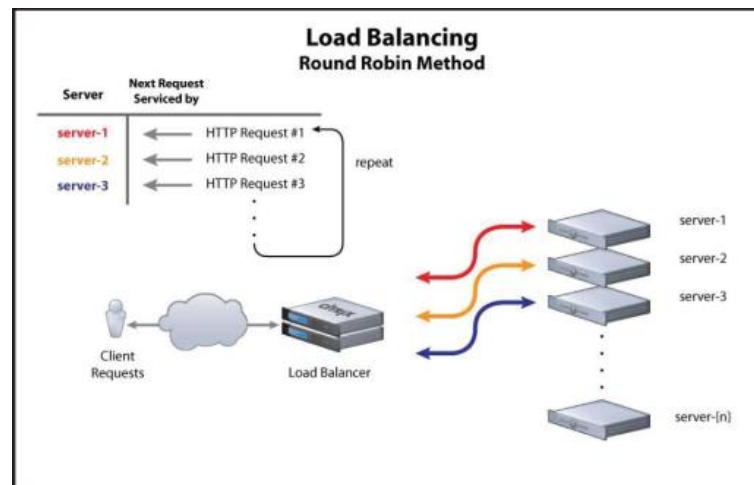
Bahasa pemrograman PHP (Hypertext Preprocessor) adalah bahasa pemrograman *scripting server-side* yang dirancang untuk pengembangan aplikasi *web*. PHP digunakan untuk memproses data formulir, membuat halaman dinamis, dan membangun situs *web* dan aplikasi *web*. PHP sering digunakan bersama dengan *database* seperti MySQL untuk membangun aplikasi *web* yang dinamis dan interaktif. PHP dapat berjalan pada berbagai sistem operasi, termasuk Windows, Linux, dan Mac OS, serta banyak platform *server web* seperti Apache dan Nginx. PHP dikembangkan pada tahun 1995 oleh Rasmus Lerdorf, dan sekarang dikelola oleh kelompok pengembang yang terorganisir dengan baik. *Interpreter* PHP dalam mengeksekusi kode PHP pada sisi *server* disebut *server side*, berbeda dengan mesin maya Java yang mengeksekusi program pada sisi klien[18]. Alasan kenapa menggunakan PHP pada penelitian ini adalah karena bahasa pemrograman ini merupakan bahasa yang digunakan dalam pembuatan *website* OpenSID.

## 9. MariaDB

MariaDB dirancang oleh mantan pengembang asli MySQL setelah terjadi akuisisi Oracle dari Sun Microsystems. Fitur utama dari MariaDB hampir sama dengan fitur pada MySQL. Para pengembang aplikasi tidak mengalami kesulitan yang nyata ketika melakukan migrasi dari MySQL ke MariaDB. Bahkan sebagian besar aplikasi untuk MySQL berfungsi dengan baik pada MariaDB tanpa memodifikasi apapun[19]. Seperti MySQL, MariaDB memuat beberapa *database* yang sudah ada yang digunakan oleh MariaDB sendiri untuk menyimpan metadata seperti informasi tentang basis data, tabel, kolom, pengguna, hak akses, *log* dan sebagainya. MariaDB adalah MySQL dengan penambahan fungsionalitas baru, bahkan segala hal mulai dari baris perintah, dokumentasi dan lainnya masih bernama MySQL. Alasan kenapa menggunakan MariaDB dalam penelitian ini adalah karena MariaDB bersifat *open-source*, artinya bisa digunakan dan dimodifikasi kode sumbernya secara gratis dan tanpa batasan. Selain itu, MariaDB memiliki kecepatan akses data yang lebih tinggi dibandingkan dengan *RDBMS* lainnya. Hal ini disebabkan oleh desain MariaDB yang dioptimalkan untuk kinerja dan menggunakan teknologi yang lebih baru dan canggih.

## 10. Algoritma Round Robin

Algoritma round robin merupakan algoritma yang paling sederhana dan paling banyak digunakan oleh perangkat *load balancing*. Algoritma round robin bekerja dengan cara membagi beban secara bergiliran dan berurutan dari satu *server* ke *server* lainnya. Konsep dasar dari algoritma round robin ini adalah dengan menggunakan *time sharing*, pada intinya algoritma ini memproses antrian secara bergiliran[20].



Gambar 10 Proses algoritma round robin

(sumber: Ellrod,[20])

Selain itu, algoritma round robin juga dapat disesuaikan dengan beberapa opsi tambahan, seperti:



a. *Weighting*

Yaitu dapat menambahkan bobot atau *weight* pada setiap *instance server* yang tersedia dalam grup. Ini dapat membantu mengatur *traffic* agar lebih merata di antara setiap *server*, terutama jika *instance server* memiliki spesifikasi atau performa yang berbeda-beda. Contohnya, ada tiga *instance server* dengan spesifikasi yang berbeda, seperti *server1* dengan *RAM 4GB*, *server2* dengan *RAM 8GB*, dan *server3* dengan *RAM 16GB*, maka kita dapat menambahkan *weight* pada *server* dengan *RAM* yang lebih besar untuk menerima lebih banyak *traffic*.

```
backend my-backend
  balance roundrobin
  server server1 192.168.0.1:80 weight 1 check
  server server2 192.168.0.2:80 weight 2 check
  server server3 192.168.0.3:80 weight 3 check
```

Gambar 11 Konfigurasi *weight* round robin

Dalam contoh konfigurasi tersebut, *server2* memiliki *weight 2*, sehingga akan menerima dua kali lipat *traffic* dibandingkan dengan *server1*, dan *server3* memiliki *weight 3*, sehingga akan menerima tiga kali lipat *traffic* dibandingkan dengan *server1*.

b. *Timeout*

Algoritma round robin dapat menambahkan opsi *timeout* pada pengaturan round robin untuk memastikan bahwa HAProxy tidak terus mengirimkan *traffic* ke *instance server* yang tidak merespon atau merespon dengan sangat lambat.

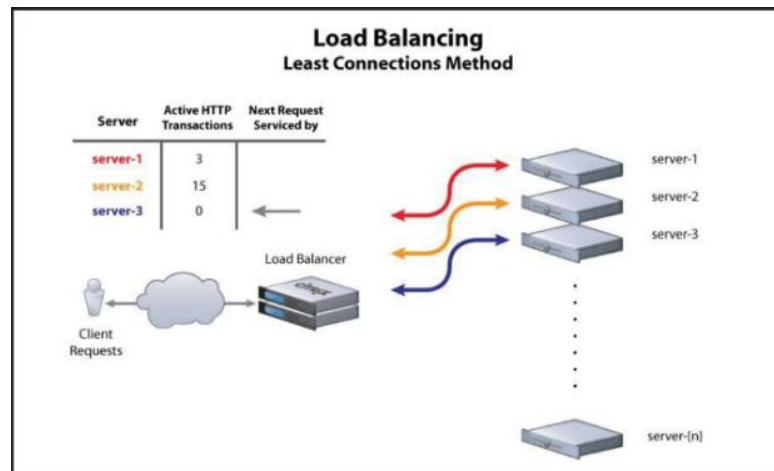
```
backend my-backend
  balance roundrobin
  timeout server 10s
  server server1 192.168.0.1:80 check
  server server2 192.168.0.2:80 check
  server server3 192.168.0.3:80 check
```

Gambar 12 Konfigurasi *timeout* round robin

Dalam contoh konfigurasi tersebut, opsi *timeout server* ditambahkan dengan nilai *10s*, yang berarti bahwa HAProxy akan menunggu hingga 10 detik untuk menerima *respons* dari setiap *instance server* sebelum mengirimkan *traffic* ke *server* lain.

## 11. Algoritma Least Connections

Algoritma least connections melakukan pembagian beban berdasarkan banyaknya koneksi yang sedang dilayani oleh sebuah *server*. *Server* dengan koneksi yang paling sedikit akan diberikan beban berikutnya, begitu pula *server* dengan koneksi banyak akan dialihkan bebannya ke *server* lain yang bebannya lebih rendah[20]. Penjadwalan ini termasuk salah satu algoritma penjadwalan dinamik, karena memerlukan perhitungan koneksi aktif untuk masing-masing *real server* secara dinamik. Metode penjadwalan ini baik digunakan untuk melancarkan pendistribusian ketika *request* yang datang sangat banyak. Sebagai contoh terdapat dua *Service-HTTP* yaitu *Service HTTP-1* (terdapat 3 *active HTTP transaction*) dan *Service HTTP 2* (terdapat 1 *active HTTP transaction*), maka *Service HTTP-2* akan menerima *request* selanjutnya dikarenakan *Service HTTP-1* > *Service HTTP-2* pada nilai transaksi aktifnya.



Gambar 13 Proses algoritma least connections

(sumber: Ellrod, [20])

Berikut adalah cara kerja algoritma least connections pada HAProxy:

- HAProxy memeriksa setiap *backend server* untuk menentukan berapa banyak koneksi aktif yang sedang berlangsung pada saat itu.
- Jumlah koneksi ini kemudian dicatat dan dibandingkan dengan jumlah koneksi pada *backend server* yang lain.
- Permintaan koneksi kemudian dikirim ke *server* dengan jumlah koneksi paling sedikit.
- Saat *server* menerima permintaan, jumlah koneksi aktifnya bertambah.
- Setelah koneksi selesai, jumlah koneksi aktif pada *server backend* akan dikurangi.

Untuk lebih memahami bagaimana algoritma least connections bekerja, perhatikan contoh berikut:

Misalkan ada tiga *server backend* dalam *cluster* yang dikonfigurasi dengan algoritma *least connections*. Saat permintaan koneksi masuk ke HAProxy, algoritma *least connections* akan memeriksa jumlah koneksi aktif di masing-masing *server backend*. Jika *server A* memiliki 10 koneksi aktif, *server B* memiliki 8 koneksi aktif, dan *server C* memiliki 6 koneksi aktif, maka permintaan koneksi akan dikirim ke *server C* yang memiliki jumlah koneksi paling sedikit. Jika beberapa permintaan koneksi masuk secara bersamaan, algoritma *least connections* akan memilih *server* dengan jumlah koneksi aktif yang paling sedikit pada saat itu. Ini membantu memastikan bahwa beban pada setiap *server backend* merata dan mencegah terjadinya *overload* pada salah satu *server*. Namun, perlu diingat bahwa algoritma *least connections* tidak mempertimbangkan faktor lain seperti beban *server* atau kecepatan respon. Jadi, jika salah satu *server backend* mengalami masalah, misalnya *overload* atau jaringan yang lambat, maka algoritma *least connections* tidak akan mengetahuinya dan terus mengirimkan permintaan koneksi ke *server* tersebut.

## 12. Algoritma Source

Dengan algoritma *source*, penyeimbang beban akan memilih *server* mana yang akan digunakan berdasarkan *hash* dari *IP* sumber permintaan, seperti alamat *IP* pengunjung. Metode ini memastikan pengguna tertentu secara konsisten terhubung ke *server* yang sama. HAProxy akan mengambil data dari *server* secara tetap, tidak berpindah-pindah. Sehingga jika sebuah sesi koneksi terjadi di *upstream* pertama, tidak akan terputus hingga sesi koneksi tersebut berakhir[21]. Algoritma *source* pada HAProxy merupakan salah satu metode *balancing* yang dapat digunakan untuk memilih *backend server* yang akan melayani permintaan dari *client*. Algoritma ini bekerja dengan memilih *backend server* berdasarkan alamat *IP* sumber (*source IP address*) dari *client* yang melakukan permintaan. Dalam penggunaan algoritma *source*, HAProxy akan mencoba untuk mempertahankan koneksi antara *client* dan *backend server* yang sama selama mungkin. Hal ini bertujuan untuk mengoptimalkan kinerja dan kecepatan akses antara *client* dan *backend server*. Namun, jika *backend server* yang sedang dipilih mengalami masalah atau tidak tersedia, maka HAProxy akan memilih *backend server* lainnya untuk memastikan ketersediaan layanan yang optimal bagi *client*.

Algoritma *source* HAProxy bekerja dengan cara memberikan prioritas pada koneksi dari *IP address* yang sama. Ketika koneksi baru masuk, HAProxy akan mengecek *IP address* pengirimnya dan mencari *backend server* yang sesuai. Apabila terdapat koneksi sebelumnya dari *IP address* yang sama, maka HAProxy akan mengarahkan koneksi baru tersebut ke *backend server* yang sama dengan koneksi sebelumnya. Dalam algoritma *source*, prioritas koneksi berdasarkan *IP address* dapat diatur dengan dua cara. Pertama, HAProxy dapat memprioritaskan koneksi dari *IP address* yang pernah terkoneksi sebelumnya

dengan *backend server* yang sama, atau kedua, HAProxy dapat memprioritaskan koneksi dari *IP address* yang pernah terkoneksi sebelumnya dengan *backend server* yang memiliki *load* yang lebih rendah.

Algoritma source pada HAProxy memiliki beberapa keunggulan, di antaranya:

- a. Meningkatkan efisiensi: Dengan mengarahkan koneksi baru ke *backend server* yang sama dengan koneksi sebelumnya, algoritma source dapat mengurangi *overhead* pada *server* dan memaksimalkan penggunaan sumber daya yang tersedia.
- b. Meningkatkan keamanan: Dalam beberapa kasus, algoritma source dapat membantu mengidentifikasi serangan *DDoS* dan mencegahnya dengan memprioritaskan koneksi dari *IP address* yang terkoneksi sebelumnya.
- c. Meningkatkan kinerja: Dengan memprioritaskan koneksi ke *backend server* yang memiliki *load* yang lebih rendah, algoritma source dapat membantu mempercepat waktu *respons* dari *server* dan memastikan kinerja yang lebih baik.

Namun, perlu diingat bahwa algoritma source tidak selalu menjadi pilihan yang terbaik dalam setiap kasus penggunaan HAProxy. Oleh karena itu, sangat penting untuk memahami kebutuhan dan tujuan dari lingkungan jaringan yang akan digunakan dan memilih algoritma yang paling sesuai untuk memaksimalkan kinerja dan keamanan jaringan.

### 13. Algoritma URI

URI (Uniform Resource Identifier) adalah *string* karakter yang digunakan untuk mengidentifikasi sumber daya yang tersedia di *Internet*. URI terdiri dari skema, otoritas, jalur, kueri, dan fragmen. Dalam HAProxy, algoritma URI digunakan untuk melakukan *routing* berdasarkan *path* pada permintaan *HTTP* yang masuk. Algoritma URI pada HAProxy memeriksa *path* pada permintaan dan membandingkannya dengan pola yang telah ditentukan. Jika pola cocok dengan *path* pada permintaan, HAProxy akan mengarahkan permintaan ke *backend* yang sesuai dengan pola yang ditemukan. Setelah HAProxy menemukan pola *path* yang cocok dengan *path* pada permintaan, maka permintaan akan diarahkan ke *backend* yang sesuai dengan pola tersebut. Algoritma URI pada HAProxy sangat berguna untuk melakukan *routing* permintaan *HTTP* ke *backend* yang tepat berdasarkan *path* yang diminta. Algoritma ini mendistribusikan beban berdasarkan *path* permintaan. Misalnya, jika ada dua *server backend* yang menyediakan layanan berbeda (contoh: */api* dan */website*), algoritma URI dapat digunakan untuk memastikan permintaan yang masuk ke *backend* yang sesuai.

Algoritma URI pada HAProxy terdiri dari beberapa tahap sebagai berikut:

- a. *Parsing URI*: HAProxy memecah URI menjadi tiga bagian utama: skema (*http* atau *https*), *domain*, dan *path*. Ini memungkinkan HAProxy untuk melakukan pemetaan permintaan ke *backend* yang sesuai.
- b. Menentukan *backend*: Berdasarkan *domain* yang diminta, HAProxy menentukan *backend* mana yang akan melayani permintaan tersebut. Ini dapat dilakukan dengan menggunakan *frontend* dan *backend* yang telah dikonfigurasi sebelumnya.
- c. Menentukan *path*: Setelah *backend* ditentukan, HAProxy mencocokkan *path* dengan aturan *routing* yang telah dikonfigurasi untuk *backend* tersebut. Aturan *routing* ini bisa berupa *pattern matching*, *regular expression*, atau *string matching*.
- d. *Load balancing*: Setelah *backend* dan *path* ditentukan, HAProxy akan memilih *server backend* yang akan melayani permintaan. Ini dilakukan dengan menggunakan algoritma *load balancing* yang telah dikonfigurasi sebelumnya.
- e. *Proxying request*: Setelah *server backend* dipilih, HAProxy akan meneruskan permintaan ke *server* tersebut dan menunggu *respons* dari *server*.
- f. Memproses *respons*: Setelah menerima *respons* dari *server backend*, HAProxy akan melakukan pemrosesan lebih lanjut, seperti *logging*, *caching*, dan kompresi.
- g. Mengirimkan *respons*: Akhirnya, HAProxy akan mengirimkan *respons* dari *server backend* ke klien yang membuat permintaan awal.

#### 14. Apache JMeter

Apache JMeter adalah perangkat lunak *open source*, 100% aplikasi Java murni dirancang untuk memuat tes perilaku fungsional dan mengukur kinerja. Ini pada awalnya dirancang untuk pengujian aplikasi *web* tetapi diperluas untuk fungsi tes lainnya[22]. Apache JMeter adalah aplikasi *open source* berbasis Java yang dapat dipergunakan untuk *performance test*. Bagi seorang QA Engineer, Apache JMeter bisa digunakan untuk melakukan *load/stress testing web application*, *FTP application* dan *database server test*. Apache JMeter dapat digunakan untuk menguji kinerja baik pada sumber daya statis dan dinamis (*web services SOAP / REST*), *web* bahasa dinamis - PHP, Java, ASP.NET, dan lain-lain. Hal ini dapat digunakan untuk mensimulasikan beban berat pada *server*, sekelompok *server*, jaringan atau objek untuk menguji kekuatan atau untuk menganalisa kinerja secara keseluruhan di bawah jenis beban yang berbeda. Alasan kenapa menggunakan *software* Apache JMeter pada penelitian ini adalah karena Apache JMeter adalah perangkat lunak sumber terbuka, sehingga dapat digunakan secara gratis dan memiliki dukungan komunitas yang luas. Kemudian karena *user friendly*, Apache

JMeter memiliki antarmuka pengguna yang mudah digunakan dan dapat dengan mudah dipahami oleh pengguna yang tidak berpengalaman. Apache JMeter mendukung berbagai protokol seperti *HTTP*, *HTTPS*, *FTP*, *JDBC*, *JMS*, *LDAP*, *TCP*, dan banyak lagi, sehingga pengguna dapat menguji berbagai jenis aplikasi.