BAB II
TINJAUAN PUSTAKA

Dalam bab ini akan dibahas mengenai *server* yaitu sistem komputer yang menyediakan sumber daya untuk pusat penyimpanan data dan layanan khusus, *virtual private server* yaitu sebuah mesin dengan kapasitas besar yang dibagi ke beberapa mesin virtual, *web server*, nginx sebagai aplikasi *web server*, *vertical scaling* yaitu upaya untuk meningkatkan sumber daya dari sebuah *server*, *horizontal scaling* yaitu upaya untuk meningkatkan jumlah *sever*, *Load balancing* yaitu metodologi jaringan komputer untuk mendistribusikan beban kerja dibeberapa komputer, HAproxy sebagai aplikasi *load balancer*, dan ApacheBench yaitu *software* yang digunakan untuk mengukur performa dari sebuah *web server*.

2.1 *Server*

Server atau dalam bahasa Indonesia biasa disebut peladen merupakan suatu sistem komputer yang menyediakan sumber daya untuk pusat penyimpanan data dan layanan khusus. Data yang disimpan melalui *server* berupa informasi dan beragam jenis dokumen yang kompleks. Secara bentuk, *server* dapat berupa *hardware, software,* atau *virtual machine*. *Server* berupa *hardware* biasanya berbentuk jaringan komputer yang memiliki ukuran sangat besar dengan menampung beberapa prosesor dan RAM yang juga berkapasitas besar.

Dilihat dari fungsi dan tujuan penggunaan *server* memiliki banyak jenis seperti *web server*, *mail server*, *database server*, *file server*, dan lain sebagainya.

2.2 *Virtual private server*

Virtual private server yaitu sebuah teknologi *server side* yang memungkinkan sebuah mesin dengan kapasitas besar dibagi ke beberapa mesin virtual. Tiap mesin virtual melayani sistem operasi dan perangkat lunak secara mandiri dan dengan konfigurasi yang cepat. Kemajuan teknologi saat ini memungkinkan untuk

mengembangkan aplikasi *remote desktop* dari sebuah *smartphone* ke sebuah mesin virtual [7].

2.3 *Web server*

Tugas utama dari *Web server* adalah menerjemahkan permintaan ke dalam respon yang cocok untuk keadaan pada saat itu, ketika *client* membuka komunikasi dengan Apache, Apache mengirimkan permintaan untuk sumber daya. Apache menyediakan sumber daya yang baik atau memberikan respon alternatif untuk menjelaskan mengapa permintaan tidak dapat terpenuhi. Dalam banyak kasus, sumber daya adalah *Hypertext Markup Language* (HTML) halaman web yang berada pada disk lokal, tetapi ini hanya pilihan sederhana. Hal ini dapat berupa file gambar, hasil dari sebuah *script* yang menghasilkan output HTML, applet Java yang diunduh dan dijalankan oleh *client*, dan seterusnya, Apache menggunakan HTTP untuk berbicara dengan *client*. Permintaan / tanggapan protokol ini, yang berarti bahwa ia mendefinisikan bagaimana membuat permintaan *client* dan bagaimana *server* menanggapi mereka. Setiap komunikasi HTTP dimulai dengan permintaan dan berakhir dengan jawaban. Executable Apache mengambil nama dari protokol, dan pada sistem Unix umumnya disebut httpd, kependekan daemon HTTP.

Cara kerja koneksi *web server* dan *client*, web atau World Wide Web (WWW) merupakan salah satu layanan Internet yang paling popular, bisa dikatakan Web adalah “wajahnya Internet”, berbagai hal dapat “ditampilkan” dihalaman Web, mulai dari teks, gambar, video, musik, dan sebagainya. Protokol bernama HTTP (Hyper Text Transfer Protocol) bertanggung jawab menangani proses komunikasi antara *client* Web (browser atau Web browser) dan *server* Web. Ketika user mengetikkan alamat *server* web atau situs pada kotak Address atau URL (Uniform Resource Locator) dan menekan [ENTER] maka Web Browser akan mencoba membentuk koneksi ke sebuah *server* Web. Browser meminta (*request*) sebuah halaman Web dan kemudian menampilkan hasilnya [8].

Bagaimana koneksi dilakukan dan apa yang terjadi selama proses dapat dijelaskan dengan sederhana sebagai berikut [8]:

1. Web browser memecah URL menjadi 3 bagian, yaitu:
   a. Bagaimana protokol (misal:http,ftp,dan sebagainya)
   b. Hostname *Server* Web
   c. Nama file home page (misal: index.html)
2. Web browser berkomunikasi dengan resolver (mencari *server* DNS), meminta informasi IP address, setelah IP address diketahui selanjutnya IP address ini digunakan untuk koneksi dengan *server* Web
3. Web browser kemudian membentuk koneksi dengan *server* menggunakan IP address dan port 80 (standar port untuk http).
4. Web browser mengirimkan *request* GET ke *server* Web, menanyakan file home page (dalam hal ini index.html).
5. *Server* Web mengirimkan halaman Web yang diminta dalam bentuk tag-tag HTML kepada Web browser.
6. Web browser membaca tag-tag HTML, kemudian menampilkan dalam keadaan sudah diformat.*Server* Web memutus koneksi dengan *client*.
7. *Server* web memutus koneksi dengan *client*.

2.4 Nginx

Nginx atau biasa disebut “Engine-x”, adalah *open source web server*. Nginx selain digunakan sebagai *web server* juga memiliki fitur untuk digunakan sebagai *reverse proxy*, HTTP *cache*, dan *load balancer*. Nginx dibuat oleh Igor Sysoev dan dirilis ke publik pada bulan Oktober 2004. Saat awal dirilis Igor meyakinkan publik bahwa Nginx dapat menjadi jawaban untuk mengatasi permasalahan yang ada pada saat itu yaitu permasalahan performa *web server* jika memiliki koneksi aktif lebih dari 10.000 koneksi secara bersamaan. Nginx menawarkan penggunaan memori yang lebih rendah dibandingkan *web server* lainnya dan juga beberapa fitur seperti: *reverse proxy*, IPv6, *Load balancing*, FastCGI *support*, *web sockets*, *handling static files*, TLS/SSL [9].

Nginx didirikan atas dasar *event-base architecture* (EBA). Komponen berinteraksi secara khusus dengan menggunakan *event notification*, bukan metode panggilan langsung. *Event Notification* ini terjadi dari tugas yang berbeda. Kemudian diantrikan untuk diproses oleh *event handler* (pengendali event). *Event Handler* berjalan secara berulang, ketika *event* tersebut diproses, kemudian dikeluarkan dari antrian dan kemudian dilanjutkan ke proses selanjutnya. Dengan demikian, pekerjaan yang dilakukan oleh sebuah *thread* (jalur) sangat mirip dengan *scheduler* (agenda), persilangan beberapa koneksi ke satu aliran eksekusi.

Dibandingkan dengan *thread-based architecture*, *event-base architecture* memberikan hasil kinerja yang lebih baik. Di *event-base architecture* ada sejumlah *thread* yang melakukan tugas dan tidak akan membentuk *thread* baru. Dengan demikian dapat mencapai utilitas CPU yang lebih baik dan *memory footprint* yang lebih baik. Tidak ada lagi koneksi *switching* yang berlebihan dan tidak perlu terjadinya tumpukan *thread* untuk setiap koneksi, CPU menjadi satu-satunya peng-hambat dari sebuah aplikasi *event-base architecture* [10].

2.5 *Vertical scaling*

Vertical scaling adalah upaya untuk meningkatkan sumber daya dari sebuah *server* seperti menambah RAM, storage, dan lain sebagainya. Pada *vertical scaling* bisa berupa hanya 1 *server*. *Vertical scaling* bisa berupa peningkatan kapasitas, penambahan perangkat, penggantian perangkat, atau bahkan penggantian *server* itu sendiri. Pada *vertical scaling* sangat sulit untuk membuat sistem tetap bisa berjalan normal selama proses *scaling*.

Beberapa layanan *database* juga menyediakan dukungan untuk melakukan *vertical scaling* seperti MySQL, MariaDB, AmazonRDS.

2.6 *Horizontal scaling*

Horizontal scaling adalah upaya untuk meningkatkan jumlah *sever*. Spesifikasi *server* yang ditingkatkan bisa berbeda dengan *server* yang sudah ada,

bisa lebih rendah ataupun lebih tinggi. Pada *horizontal scaling* akan lebih mudah untuk membuat sistem tetap berjalan normal selama proses *scaling* dibanding dengan *vertical scaling*. Hal ini dikarenakan proses *scaling* tidak mengganggu *server* yang sedang berjalan.

Beberapa layanan *database* yang menyediakan dukungan untuk melakukan *horizontal scaling* adalah MongoDB, Cassandra.

2.7 *Load balancing*

Load balancing merupakan metodologi jaringan komputer yang bekerja dengan cara mendistribusikan beban kerja di beberapa komputer atau kluster komputer untuk mencapai pemamfaatan optimal dari sumber daya, memaksimalkan *throughput*, meminimalkan waktu respon, dan menghindari kelebihan beban. Kluster *server* sendiri merupakan gabungan beberapa perangkat komputer yang saling terhubung dan bekerjasama sehingga mereka dapat dilihat sebagai satu sistem dalam banyak aspek, dan kluster komputer biasanya digunakan untuk meningkatkan kinerja dan ketersediaan atas satu komputer. *Load balancer* dapat melakukan berbagai fungsi seperti peyeimbang beban, *traffic engineering*, dan pemindahan jalur lalu lintas. *Load balancer* dapat melakukan pemeriksaan kesehatan pada *server*, aplikasi, dan konten untuk meningkatkan ketersediaan layanan dan pengelolaanya. Dalam penerapan *load balancer* ada berbagai algoritma penjadwalan yang dapat digunakan untuk mendistribusikan beban kepada setiap *server* didalam sekumpulan *server* [5]. Algoritma penjadwalan yang umumnya digunakan adalah:

- *Round robin*

Untuk model penjadwalan tipe *round robin*, *load balancer* mendistribusikan permintaan layanan sama rata ke seluruh real *server* tanpa memperdulikan kapasitas *server* atau pun beban permintaan layanan. Jika ada empat *server* (A,B,C,D), maka permintaan layanan 1 akan diberikan *load balancer* kepada *server* A, permintaan layanan 2 ke *server* B, permintaan layanan 3 ke *server* C, permintaan layanan 4 ke

server D dan permintaan layanan 4 akan kembali ke *server* A. Mekanisme ini dapat dilakukan jika seluruh *server* menggunakan spesifikasi komputer yang sama. Konsep dasar dari algoritma ini adalah dengan menggunakan *time sharing*. Pada dasarnya algoritma ini sama dengan FCFS, hanya saja bersifat *preemptive*. Setiap proses mendapatkan waktu CPU yang disebut dengan waktu quantum (*quantum time*) untuk membatasi waktu proses, biasanya 1-100 milidetik. Setelah waktu habis, proses ditunda dan ditambahkan pada *ready queue*. Algoritma *roundrobin* merupakan algoritma yang paling sederhana dan banyak digunakan oleh perangkat *load balancing*. Algoritma ini membagi beban secara bergiliran dan berurutan dari satu *server* ke *server* lain sehingga membentuk putaran.

Penjadwalan ini merupakan:

- Penjadwalan preemptive, bukan di-preempt oleh proses lain, tapi terutama oleh penjadwal berdasarkan lama waktu berjalannya proses, disebut preempt by time.
- Penjadwal tanpa prioritas. Semua proses dianggap penting dan diberi sejumlah waktu proses yang disebut kwanta (quantum) atau time slice dimana proses itu berjalan.

Ketentuan algoritma *roundrobin* adalah sebagai berikut:

- Jika kwanta dan proses belum selesai maka proses menjadi runnable dan pemroses dialihkan ke proses lain.
- Jika kwanta belum habis dan proses menunggu suatu kejadian (selesainya operasi I/O), maka proses menjadi blocked dan pemroses dialihkan ke proses lain.
- Jika kwanta belum habis tapi proses telah selesai, maka proses diakhiri dan pemroses dialihkan ke proses lain. Algoritma *round robin* menggilir proses yang ada di antrian. Proses akan mendapat jatah sebesar quantum time. Jika quantum time-nya habis atau proses sudah selesai, CPU akan dialokasikan ke proses berikutnya. Tentu proses ini cukup adil karena tak

ada proses yang diprioritaskan, semua proses mendapat jatah waktu yang sama dari CPU yaitu (1/n), dan tak akan menunggu lebih lama dari (n-1)q dengan q adalah lama 1 quantum. Algoritma ini sepenuhnya bergantung besarnya time quantum. Jika terlalu besar, algoritma ini akan sama saja dengan algoritma first come first served. Jika terlalu kecil, akan semakin banyak peralihan proses sehingga banyak waktu terbuang. Permasalahan utama pada *round robin* adalah menentukan besarnya time quantum. Jika time quantum yang ditentukan terlalu kecil, maka sebagian besar proses tidak akan selesai dalam 1 quantum. Hal ini tidak baik karena akan terjadi banyak switch, padahal CPU memerlukan waktu untuk beralih dari suatu proses ke proses lain (disebut dengan context switches time). Sebaliknya, jika time quantum terlalu besar, algoritma *Roundrobin* akan berjalan seperti algoritma first come first served. Time quantum yang ideal adalah jika 80% dari total proses memiliki CPU burst time yang lebih kecil dari 1 time quantum[5].

- *Least connection*

Algoritma ini melibatkan pengiriman permintaan layanan baru ke *server* dengan jumlah koneksi bersamaan. Metode ini memerlukan penyeimbang beban untuk melacak jumlah koneksi aktif secara bersamaan pada setiap *server* dan mengirim permintaan ke *server* dengan sedikitnya jumlah koneksi yang aktif bersamaan.

- *Source*

Algoritma *source* bekerja dengan cara meng-hash alamat IP sumber lalu merutekan ke *server* aplikasi sesuai bobotnya. Hal ini memastikan bahwa *request* dari alamat IP yang sama akan selalu dilayani oleh *server* yang sama juga

2.8 HAproxy

Haproxy adalah produk *opensource* yang mendukung keperluan penyeimbang beban dan failover *web server*, banyak digunakan untuk keperluan reverse proxy di site-site yang trafik hariannya tinggi. Pada Haproxy memiliki beberapa parameter di dalamnya yaitu sebagai berikut [6].

- Global parameter : berisi parameter manajemen proses dan keamanan, tuning kinerja, debugging, dan userlist
- Proxy
- Konfigurasi *Server*
- Manipulasi HTTP
- Accesslist
- Logging
- Statistik dan monitoring

2.9 ApacheBench

ApacheBench adalah software berbasis perintah CLI yang digunakan untuk mengukur performa dari sebuah *web server*. Software ini dapat melakukan pengetesan terhadap sebuah *web server* dengan melakukan banyak *request* sekaligus ataupun secara bergantian. Hasilnya akan keluar setelah pengetesan selesai.