Coding : Jurnal Komputer dan Aplikasi
Volume 09, No. 02 (2021), hal 316-326 ISSN : 2338-493X

IMPLEMENTASI TEKNIK *SCALING* PADA SISTEM MANAJEMEN *BALANCING* SERVER BERBASIS WEBSITE

Ridho Subhi[1], Ikhwan Ruslianto[2], Uray Ristian[3]
[1,2,3]Jurusan Rekayasa Sistem Komputer, Fakultas MIPA Universitas Tanjungpura
Jalan Prof. Dr. H. Hadari Nawawi Pontianak
Telp./Fax : (0561) 577963
e-mail: [1]ridho.subhi@student.untan.ac.id, [2]ikhwanruslianto@siskom.untan.ac.id,
[3]eristian@siskom.untan.ac.id

Abstrak

Peningkatan jumlah pengguna internet dapat menyebabkan jumlah pengunjung *website* meningkat. Meningkatnya jumlah pengunjung *website* mengakibatkan kinerja server tidak optimal dalam menyediakan sumber daya untuk menerima *request*. Salah satu permasalahan yang akan terjadi adalah server dapat menjadi *down*. Solusi dari permasalahan tersebut adalah dengan menerapkan teknik *scaling* untuk mengalihkan *request* yang terjadi diserver. Dengan melakukan *scaling*, server dapat mengontrol *storage* ketika terjadi *request* dengan jumlah tinggi sehingga sistem dapat melakukan *balancing* terhadap server. Pada penelitian ini, server yang digunakan adalah *elastice compute* (EC2) dan berjumlah 2 server. Teknik *scaling* dilakukan untuk mengelola jaringan server menggunakan konsep horizontal *scaling* dengan parameter CPU dan memori. Sistem menggunakan *tools httperf* untuk melakukan *request* dan *iptables* untuk melakukan *reject* semua koneksi *request* dari protokol ICMP pada server. Penolakan *request* pada server menyebabkan penurunan penggunaan CPU dan memori. *Output* yang dihasilkan mencakup tiga data yaitu CPU, memori serta *response time* dengan *scaling* dan tanpa *scaling* server. Hasil akhir menunjukkan bahwa *request* yang dilakukan *reject* dapat berpengaruh pada penggunaan CPU dan memori dengan rata-rata nilai selisih tiap penurunan adalah 6,07% perdetik dan 2,9 MB perdetik. Nilai rata-rata *response time* tanpa *scaling* adalah 564,4 ms. Sedangkan *response time* dengan *scaling* tidak memiliki nilai *request* karena koneksi ditolak oleh server.

Kata kunci: *Scaling*, *Balancing* Server, *Elastic Compute* (EC2), Manajemen Jaringan

1 PENDAHULUAN

Perkembangan jaringan internet banyak mengalami kemajuan, terutama pada jaringan berbasis publik. Kemajuan pada perkembangan jaringan internet menyebabkan jumlah pengguna internet semakin bertambah. Bertambahnya pengguna internet akan berdampak pada server yang menjadi pusat penyimpanan data. Pada umumnya server akan terus menerima paket yang masuk dari *client* untuk tetap dapat membuat koneksi dengan *client* tanpa memperhatikan kondisi sumber daya *storage* pada server. Jika server tidak dapat mengelola sumber daya dengan baik, maka server akan *down* dan tidak dapat melayani *request* koneksi yang masuk. Hal tersebut dapat menyebabkan pengguna internet tidak dapat mengakses seluruh data maupun aplikasi *website* yang ada pada jaringan server. Dengan dampak tersebut server harus dapat mengelola dan mengoptimalkan kinerja sumber daya untuk melayani *request* koneksi yang terjadi, agar koneksi jaringan dapat bekerja dengan optimal.

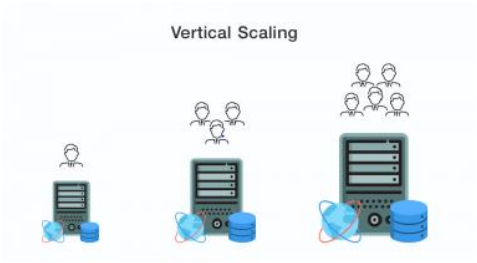
Berdasarkan pemasalahan tersebut, maka dilakukan penelitian dengan Judul "Implementasi Teknik *Scaling* Pada Sistem Manajemen *Balancing* Server Berbasis *Website*". Implementasi teknik *scaling* diterapkan menggunakan *tools iptables* untuk menolak semua koneksi dari protokol ICMP ketika penggunaan *storage* server telah melebihi 90%. Sistem akan menampilkan status server pada aplikasi sehingga

memudahkan administrator dalam melihat dan mengontrol *storage* server. *Scaling* pada penelitian ini dilakukan dengan manual menggunakan *button* khusus pada *website*. Adanya sistem ini diharapkan dapat membantu administrator dalam mengelola kondisi server.

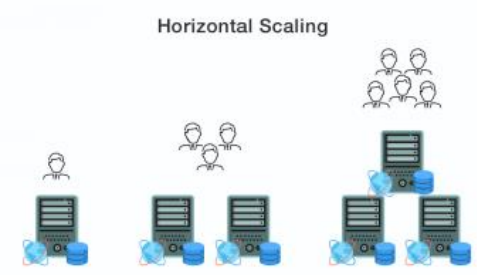
2 LANDASAN TEORI

2.1 *Scaling* Server

Scaling merupakan kemampuan sistem untuk menyesuaikan sumber daya yang dimiliki seperti menurunkan atau menambah jumlah proses sesuai kebutuhan sistem tanpa mengganggu proses yang sedang berjalan[1]. Sedangkan *autoscaling* merupakan suatu teknik yang digunakan dalam *cloud computing* untuk mengelola *server* komputasi secara otomatis, sehingga administrator jaringan tidak perlu melakukan penambahan ataupun pengurangan *server* secara manual ketika menjalankan suatu layanan *cloud*[2]. Sebagai contoh, *scaling* dapat menyesuaikan kapasitas server atau mesin virtual (VM) sesuai dengan lonjakan permintaan (*request*) atau aktivitas. Jika terjadi lonjakan permintaan atau aktivitas maka kapasitas server akan ditingkatkan secara otomatis, begitu pula sebaliknya. Terdapat 2 macam *scaling* antara lain adalah *vertical scaling* dan horizontal *scaling* yang dapat dilihat pada Gambar 1 dan 2.



Gambar 1 *Vertical Scaling*



Gambar 2 Horizontal *Scaling*

Perbedaan dari *Vertical* dan Horizontal *scaling* adalah *vertical scaling* melakukan penskalaan sumber daya server dalam satu server tunggal. Sedangkan horizontal *scaling* melakukan penskalaan sumber daya server dengan menambah lebih banyak sumber daya pada satu server didalam satu jaringan. Pada penelitian, metode yang digunakan adalah horizontal *scaling*.

2.2 *Cloud Computing*

Cloud computing merupakan satu set data server berskala besar yang menyediakan komputasi dan layanan penyimpanan yang besar secara stabil kepada pengguna[3]. Dengan adanya *cloud computing*, semua media komputasi dapat terhubung dan digunakan dengan mudah dan fleksibel. *Cloud computing* juga menghemat biaya yang dikeluarkan dalam membangun suatu arsitektur jaringan. Pada penelitian ini, *cloud computing* digunakan sebagai konsep dasar dari arsitektur jaringan yang akan dibangun. Model dari *cloud computing* yang digunakan adalah *public cloud*. *Cloud computing* digunakan untuk membangun seluruh arsitektur jaringan yang digunakan pada penelitian.

2.3 Manajemen Jaringan

Manajemen jaringan merupakan suatu usaha untuk memelihara seluruh sumber daya jaringan dalam keadaan baik termasuk operasional, administrasi, pemeliharaan dan penyediaan layanan jaringan[4]. Karena saat ini jaringan sangat kompleks, arsitektur jaringan harus terdiri dari komponen-komponen peralatan jaringan yang baik dalam perawatannya agar menghasilkan koneksi yang baik untuk data *request* yang akan diakses oleh *client*. Manajemen jaringan sangat diperlukan untuk mengelola peralatan tersebut.

2.4 *Stress Testing*

Testing atau ujicoba adalah proses menjalankan sebuah aplikasi dengan tujuan menemukan sebuah permasalahan. *Stress testing* adalah bagian dari *performance testing*. *Performance testing* dilakukan dengan cara melakukan permintaan dalam jumlah besar, seperti mengakses sistem dengan banyak *user* dalam waktu yang bersamaan. *Performance testing* memiliki tujuan untuk mengevaluasi kemampuan dari suatu sistem dalam menangani sebuah permintaan atau *request*.

Adapun tujuan lain dari *stress testing* dan *performance testing* adalah untuk mengetahui *response time testing* dan penggunaan sumber daya[5]. Pada penelitian ini *stress testing* digunakan untuk pengumpulan data *request* ke server sebagai dasar acuan dari *rule* yang akan dibangun. *Stress testing* juga digunakan pada saat dilakukan skenario pengujian aplikasi *website* monitoring *storage*. *Stress testing* akan digunakan sebagai pengujian untuk mengetahui performa dari server ketika menerima *traffict request* yang tinggi.

2.5 CPU *Usage*

CPU *usage* adalah grafik yang menunjukkan angka sumber daya yang dibutuhkan sebuah *prosessor* pada *Central Processing Unit* (CPU) dalam bekerja menangani proses dan satuannya ditampilkan dalam hitungan persen[6]. Jika semakin kecil pemakaian CPU pada sebuah server maka semakin kecil juga kemungkinan untuk terjadi *overload task*. Server yang tidak mengalami *overload* berarti dalam kondisi optimal dan baik. Pada penelitian ini CPU *usage* akan digunakan sebagai salah satu parameter *rule*. Parameter lain yang digunakan pada *rule* adalah memori dan *diskspace*.

2.6 *Response Time Testing*

Response time testing adalah pengujian waktu respons yang mengacu pada waktu yang diperlukan untuk satu *node* sistem dalam menanggapi setiap kinerja transaksi atau permintaan[7] . Waktu respon dimulai ketika pengguna mengirimkan permintaan atau *request* dan berakhir ketika tidak terjadi koneksi. Pada penelitian ini *response time testing* dilakukan untuk mengetahui data dari waktu tanggap ketika *client* melakukan sejumlah *request* pada saat dilakukan pengujian kinerja *storage* server.

2.7 *Virtual Private Server*

VPS (*Virtual Private Server*) merupakan teknologi virtualisasi sebuah *physical* server yang dibagi menjadi beberapa *virtual private* server sehingga setiap server terlihat bekerja seperti sebuah server mandiri[8]. Setiap VPS memiliki *full root acces*, sistem operasi, CPU dan *disk* yang berdiri sendiri. Berbeda dengan *shared hosting* yang menggunakan server secara bersamaan dan saling mempengaruhi satu sama lain. Proses yang berjalan pada satu *vps* tidak akan mempengaruhi *vps* lain dalam sebuah arsitektur jaringan.

Pada penelitian ini digunakan sebuah VPS yang terkoneksi secara *cloud* dan terhubung dengan *website* Berkahbarang yang sudah di-*deploy* pada server. Server yang digunakan memiliki beberapa layanan protokol seperti *secure shell* (SSH), *hypertext transfer protocol* (HTTP) dan *internet control message protocol* (ICMP). Protokol SSH akan digunakan sebagai protokol yang dapat me-*remote* server dari jarak jauh untuk melakukan berbagai instalasi dan konfigurasi kebutuhan *tools* dari penelitian ini. Protokol HTTP berfungsi sebagai komunikasi data antara *client* dengan *webserver*. Sedangkan protokol ICMP akan digunakan untuk mengetahui kesalahan dari koneksi server.

2.8 *Secure Shell* (SSH)

Secure Shell atau SSH adalah protokol jaringan yang berada di lapisan aplikasi pada protokol *tcp*/*ip* dan memfasilitasi sistem komunikasi yang aman diantara dua sistem yang menggunakan arsitektur *client* server dengan menyediakan kerahasiaan dan integritas data melalui teknik enkripsi dan dekripsi yang dilakukan secara otomatis didalam koneksinya[9]. SSH biasa digunakan untuk melakukan *remote* server jarak jauh agar administrator jaringan tidak perlu mengakses fisik server secara langsung. SSH mendukung akses untuk mengendalikan server sesuai dengan otoritas *user* yang *login*. Untuk dapat mengendalikan server secara penuh tentu *user* harus *login* sebagai *root* atau sebagai *user* dengan level yang sama dengan *root*. Pada sisi server, SSH pada umumnya dijalankan menggunakan sebuah *tools* yang dapat menerima permintaan akses dari *client* ke server. SSH juga dijalankan menggunakan sebuah *tools* untuk mendekripsi dan mengenkripsi data yang diterima dan dikirim. Salah satu contoh dari *tools* tersebut adalah *openssh*, *putty*, *winscp* dan *xshell*.

Pada penelitian ini, *port* SSH digunakan sebagai akses masuk ke server untuk melakukan konfigurasi kebutuhan *tools* server yang akan digunakan. Untuk memudahkan

administrator jaringan mengakses *port* SSH ke dalam server.

2.9 *Internet Control Message Protocol*

Internet control message protocol atau ICMP merupakan suatu protokol yang digunakan untuk melakukan tes koneksi dari sebuah *host* ke *host* lain dengan mengirimkan sebuah *request packet* ke *host* tujuan menggunakan *ip address*[10]. Terdapat dua tipe pesan yang dihasilkan oleh ICMP yaitu *error message* dan *query message*. *Error message* dihasilkan jika terjadi kesalahan pada jaringan yang sedang berjalan. Sedangkan *query message* dihasilkan jika pengirim paket mengirimkan informasi tertentu yang berkaitan dengan kondisi jaringan.

Pada penelitian ini protokol ICMP digunakan sebagai jalur komunikasi jaringan antar *client* dan server. ICMP juga digunakan untuk mengetahui keberhasilan koneksi yang terjadi pada saat melakukan konfigurasi *amazon elastic compute* (EC2), *hosting website* Berkahbarang ke server dan saat melakukan *deploymenti* aplikasi *website* monitoring ke server.

2.10 *Iptables*

Iptables adalah kelompok arsitektur pemrosesan paket jaringan aturan ke dalam tabel berdasarkan fungsi (*filter* paket, jaringan terjemahan alamat, dan paket lainnya) yang masing-masing memiliki rantai (urutan) aturan pemrosesan[11]. Aturan terdiri dari kecocokan (digunakan untuk menentukan apa yang akan dilakukan dengan pencocokan paket). Iptables mendedikasikan lima *rule* di dalam jalur pemrosesan paket seperti *prerouting, input, forward, postrouting* dan *output*. Setiap *rule* mempunyai fungsi untuk mempengaruhi atau memantau aliran paket yang masuk. Iptables dibagikan ke 3 kategori seperti:

1. *Drop*, membiarkan paket tersebut seolah-olah tidak pernah diterima.
2. *Accept*, menerima paket tersebut untuk diproses lebih lanjut.
3. *Reject*, menolak dan memberitahukan pengirim bahwa paket data tidak bisa diterima.

Iptables pada penelitian ini akan digunakan untuk melakukan *reject* koneksi ke *ip* server utama dan melakukan *accept* koneksi kembali ke *ip* server cadangan pada kondisi yang telah ditentukan.

2.11 *Httperf*

Httperf adalah *tools benchmark* yang berfungsi untuk mengukur performansi dari server. *Tools* ini menyediakan fitur yang fleksibel dalam pembuatan beban kerja sesuai dengan parameter yang diberikan. *Httperf* dapat memberikan sejumlah paket *request* dan mendukung HTTP/1.0 dan HTTP/1.1[12]. Pada penelitian ini *httperf* digunakan pada tahap pengujian untuk melakukan sejumlah *request* ke server dan akan dilihat performansi dari server tersebut

2.12 *Laravel*

Framework Laravel adalah salah satu contoh *framework* yang dapat digunakan untuk membangun aplikasi *website*, yang membuat *framework* ini banyak digunakan adalah *syntax* dari *framework* ini yang ekspresif, rapi, dan mudah dipahami hingga mempercepat proses pembuatan *website*. Laravel adalah salah satu *framework* PHP terbaik yang dikembangkan oleh Taylor Otwell. Sebagai sebuah *framework* PHP, Laravel hadir sebagai platform *web development* yang bersifat *open source*[13]. Laravel memiliki beberapa keunggulan dibanding *framework* yang lain, yaitu:

1. Menggunakan *Command Line Interface* (CLI) Artisan.
2. Menggunakan *package* manager PHP Composer.
3. Penulisan Kode Program lebih singkat, mudah dimengerti, dan ekspresif.

Selain keunggulan dibanding *framework* lainnya, Laravel juga memiliki berbagai fitur yang memudahkan pengguna dalam membangun website sesuai keperluan. *Framework* laravel mempunyai beberapa fitur yang menjadi unggulan seperti *blade*, *migration*, *eloquent* dan *server controller*. Berdasarkan fitur yang disediakan, penelitian ini akan digunakan *framework* laravel dalam membangun aplikasi *website* monitoring *storage* server. Aplikasi *website* yang dibangun akan digunakan untuk menampilkan informasi CPU, memori, *diskspace* dan aktivitas performansi CPU antar server saat sistem *scaling* bekerja dalam bentuk grafik.

2.13 *Mean*

Mean atau rata-rata adalah nilai yang mewakili sifat tengah atau posisi pusat dari suatu kumpulan nilai data[14]. *Mean* akan digunakan untuk menghitung nilai rata-rata penggunaan CPU, memori dan *response time* dari server. Rumus untuk menghitung nilai *mean* dapat dilihat pada persamaan 1 berikut.

$$\bar{x} = \frac{\sum_{i=1}^{n} xi}{n} \quad (1)$$

Keterangan:
$\bar{x}$ = *Mean* atau nilai rata-rata
n = banyaknya data *x* dalam suatu sampel
xi = nilai dari data ke-*i*

3 METODE PENELITIAN

Penelitian implementasi teknik *scaling* pada server berbasis *website* dilakukan dengan berbagai tahapan seperti studi literatur, metode pengumpulan data, analisis kebutuhan, perancangan sistem, implementasi dan pengujian.

3.1 Studi Literatur

Tahap ini dilakukan untuk pengumpulan bahan-bahan referensi dalam penelitian. Literatur referensi digunakan sebagai acuan untuk memberikan penjelasan serta melakukan tinjauan pada teori pada penelitian sebelumnya. Bentuk studi literatur yang dilakukan adalah pengumpulan berupa jurnal ilmiah, buku, dan referensi lain untuk selanjutnya dipelajari.

3.2 Metode Pengumpulan Data

Metode pengumpulan data yang dilakukan pada penelitian ini adalah dengan melakukan observasi pada referensi yang dikumpulkan dan berkaitan dengan teori mengenai *cloud computing*, *scaling* server, aplikasi *website*, server, *framework* Laravel, *pusher*, *composer*, *Iptables* dan berbagai sumber referensi lain seperti jurnal, halaman *website* dan sumber informasi lain yang mendukung penelitian ini.

3.3 Analisis Kebutuhan

Analisis kebutuhan dilakukan untuk mengetahui kebutuhan perangkat dari sistem yang dibuat. Analisis kebutuhan pada penelitian ini terdiri dari kebutuhan perangkat keras dan kebutuhan perangkat lunak.

3.3.1 Kebutuhan Perangkat Lunak

Secara umum perangkat lunak yang dibutuhkan adalah sebagai berikut:

1. *Google Chrome*
2. Platform *Amazon Elastic Compute (EC2)*
3. Xshell 6
4. OS *Ubuntu* server.
5. *Secure Shell* (SSH).
6. *Iptables*
7. LAMP
8. *Framework* Laravel
9. *Composer*

3.3.2 Kebutuhan Perangkat Keras

Secara umum perangkat keras yang dibutuhkan adalah sebagai berikut:

1. Laptop OS Linux, RAM 2GB yang digunakan sebagai *client* dan teknisi server.
2. Laptop OS Windows 10, RAM 2GB yang digunakan sebagai administrator server.

3.4 Perancangan Sistem

Pada tahap ini peneliti merancang sistem pada server agar sesuai dengan kebutuhan sistem. Sistem yang dibangun berbasis aplikasi *website*. Aplikasi akan dirancang untuk memonitoring *storage* pada dua server dan dapat melakukan pengalihan jalur koneksi ke server cadangan berdasarkan *rule* yang dibuat. Sistem yang akan dibangun berbasis aplikasi *website*. Aplikasi *website* akan di pasang pada server utama. Aplikasi yang dibangun dapat digunakan oleh administrator server. Perancangan sistem terdiri dari perancangan arsitektur sistem, perancangan perangkat lunak, perancangan aplikasi, perancangan pola *request*, perancangan *rule*, perancangan *database*, perancangan antarmuka aplikasi dan perancangan skenario pengujian. Setelah perancangan sistem dilakukan, selanjutnya memasuki tahap implementasi sistem.

3.5 Implementasi

Tahap implementasi merupakan tahapan penerapan dari perancangan sistem yang telah dibuat. Tahapan implementasi meliputi konfigurasi *amazon elastic compute* (EC2) sebagai server *virtual* dan *deployment* aplikasi *website* Berkahbarang pada server virtual, pembuatan aplikasi *website* monitoring, mengintegrasikannya dengan *Iptables* dan *rule* yang telah dibuat untuk membuat fungsi *scaling* pada server. Dalam tahap implementasi aplikasi terdiri dari implementasi sistem, implementasi kode program, implementasi *database* aplikasi dan implementasi aplikasi *website*.

3.6 Pengujian

Pengujian sistem dilakukan untuk mengetahui keberhasilan dari tahap perancangan sistem dan implementasi sistem serta untuk mendapatkan data yang akan menjadi bahan kesimpulan dari penelitian. Adapun pengujian sistem terdiri dari pengujian keberhasilan monitoring, pengujian keberhasilan *iptables* dan pengujian skenario *request*.

4 HASIL DAN PEMBAHASAN

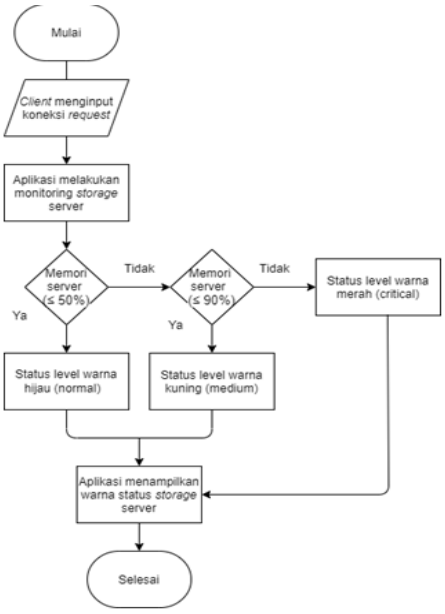
Sebelum dilakukan tahap pengujian, tahapan yang harus dilakukan adalah tahap implementasi. Tahapan implementasi meliputi konfigurasi *amazon elastic compute* (EC2) sebagai server *virtual* dan *deployment* aplikasi *website* Berkahbarang pada server virtual, pembuatan aplikasi *website* monitoring, mengintegrasikannya dengan *Iptables* dan *rule* yang telah dibuat untuk membuat fungsi *scaling* pada server. Adapun tahapan implementasi secara umum dalam implementasi aplikasi monitoring storage adalah sebagai berikut:

1. Implementasi sistem
2. Implementasi kode program
3. Implementasi *database*
4. Implementasi aplikasi *website*

Setelah dilakukan implementasi selanjutnya adalah tahap pengujian. Adapun tahap pengujian secara umum adalah sebagai berikut:

1. Pengujian keberhasilan monitoring
2. Pengujian keberhasilan *iptables*
3. Pengujian skenario *request*

Selanjutnya dilakukan implementasi *rule* untuk membuat sistem kerja aplikasi dalam memonitoring status server. Adapun kinerja *rule* dapat dilihat pada Gambar 3.



Gambar 3 Diagram Alir *Rule*

4.1 Pengujian Keberhasilan Monitoring

Pengujian keberhasilan monitoring dilakukan untuk menguji tingkat keberhasilan aplikasi dalam menampilkan status monitoring *storage* server. pada tahap ini pengujian terbagi menjadi 3 yaitu pengujian dengan sample 25 *request*, 50 *request* dan 100 *request*.

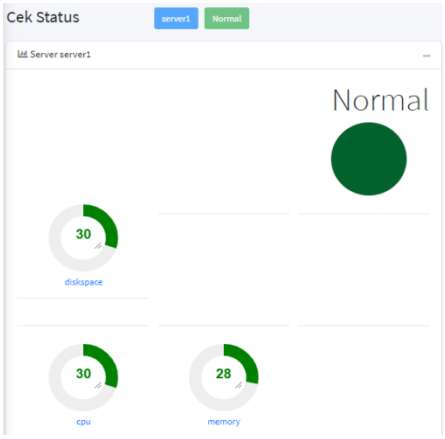
4.1.1 Pengujian 25 *Request*

Pada pengujian, dilakukan pengujian *request* dalam bentuk *ping* melalui *command prompt*. Berikut merupakan *sample* data yang didapat dari 25 *request* perdetik selama 6 detik yang dapat dilihat pada Tabel 1.

Tabel 1 Data *Sample* 25 *Request*

No.	*Request*/detik	CPU (%)	Memori (Mb)	Status server di aplikasi
1	25	0,7%	275	Normal
2	25	15,2%	278	Normal
3	25	33,5%	280	Normal
4	25	**48,3%**	**285**	Normal
5	25	30,9%	285	Normal
6	25	25,3%	285	Normal

Berdasarkan Tabel 1 dapat dilihat bahwa data penggunaan CPU dan memori tertinggi adalah 48,3% dan 285MB. Pengujian dilakukan dengan waktu yang berurutan agar dapat dilihat kenaikan memori yang dihasilkan. Berikut adalah tampilan status server pada aplikasi dapat dilihat pada Gambar 4.



Gambar 4 Tampilan Data *Sample* 1

Berdasarkan Gambar 4 dapat dilihat aplikasi berhasil menampilkan status hijau atau normal dari hasil *sample* data 25 *request*. Aplikasi dapat menampilkan status hijau karena penggunaan CPU tidak melebihi dari 50%. Berdasarkan *rule* yang dibuat jika penggunaan CPU kurang dari 50% maka aplikasi menampilkan status hijau atau normal.

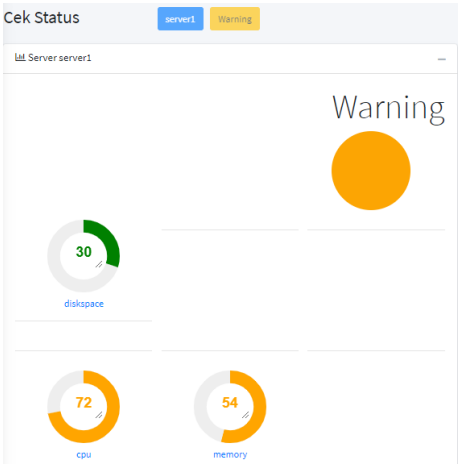
4.1.2 Pengujian 50 *Request*

Pada pengujian, dilakukan pengujian *request* dalam bentuk *ping* melalui *command prompt*. Berikut merupakan sample data yang didapat dari 50 *request* perdetik selama 6 detik pada server yang dapat dilihat pada Tabel 2.

Tabel 2 Data Sample 50 *Request*

No.	*Request*/detik	CPU (%)	Memori (Mb)	Status server di aplikasi
1	50	0,7%	275	Normal
2	50	11,4%	283	Normal
3	50	23,2%	305	Normal
4	50	54,3%	318	Medium
5	50	65,7%	327	Medium
6	50	**73,0%**	**338**	Medium

Berdasarkan Tabel 2 dapat dilihat bahwa data penggunaan CPU dan memori tertinggi adalah 73,0% dan 338MB. Pengujian dilakukan dengan waktu yang berurutan agar dapat dilihat kenaikan memori yang dihasilkan. Berdasarkan Gambar 5 dapat dilihat bahwa aplikasi berhasil menampilkan status kuning atau *warning* dari hasil *sample* data 50 *request*. Aplikasi dapat menampilkan status kuning atau *warning* dikarenakan penggunaan CPU telah melebihi 50% sehingga menyebabkan *rule* yang berjalan dapat mengubah status dari normal menjadi *warning*. Tampilan status server pada aplikasi dapat dilihat pada Gambar 5.



Gambar 5 Tampilan Data *Sample* 2

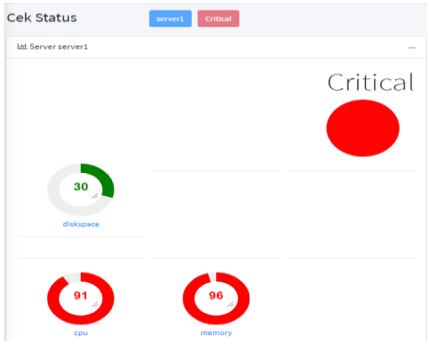
4.1.3 Pengujian 100 *Request*

Pada pengujian, dilakukan pengujian *request* dalam bentuk *ping* melalui *command prompt*. Selanjutnya dilakukan pengujian dengan 100 *request* ke server. Berikut merupakan *sample* data yang didapat dari 100 *request* perdetik selama 6 detik yang dapat dilihat pada Tabel 3.

Tabel 3 Data *Sample* 100 *Request*

No.	*Request*/detik	CPU (%)	Memori (Mb)	Status server di aplikasi
1	100	0,7%	275	Normal
2	100	25,6%	280	Normal
3	100	91,4%	299	*Critical*
4	100	100%	342	*Critical*
5	100	100%	398	*Critical*
6	100	**100%**	**470**	*Critical*

Berdasarkan Tabel 3 dapat dilihat bahwa penggunaan CPU dan memori tertinggi adalah 100% dan 470MB. Pengujian dilakukan dengan waktu yang berurutan agar dapat dilihat kenaikan memori yang dihasilkan. Berikut merupakan tampilan status server yang dapat dilihat pada Gambar 6.

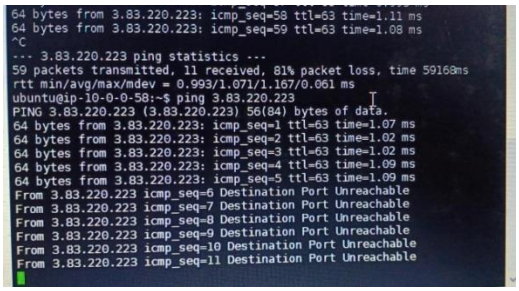


Gambar 6 Tampilan Data *Sample* 3

Berdasarkan Gambar 6 dapat dilihat bahwa aplikasi telah berhasil menampilkan status merah atau *critical* dari hasil *sample* data 100 *request*.

4.2 Pengujian Keberhasilan *Iptables*

Pengujian keberhasilan *iptables* dilakukan dengan tujuan untuk melihat berhasil tidaknya *iptables* melakukan *reject* ketika status memori *critical*. Pada Gambar 6 dapat dilihat bahwa *iptables* telah berhasil melakukan *reject* paket *request* yang terjadi di server. *Reject* dilakukan apabila penggunaan CPU telah melebihi 90%. Adapun keberhasilan *iptables* melakukan *reject* dapat dilihat pada Gambar 7.



Gambar 7 Hasil *Reject* Koneksi *Iptables*

Setelah berhasil melakukan *reject* koneksi selanjutnya dibuat Tabel hasil pengujian untuk melihat penurunan dari penggunaan CPU dan memori yang dapat dilihat pada Tabel 4.

Tabel 4 Hasil Pengujian *Iptables*

No.	*Request*/detik	Waktu (detik)	CPU	Mem	Server 1	Server 2
1	70	1	0,8%	750	Accept	Reject
2	70	1	30,5%	765	Accept	Reject
3	70	1	48,2%	781	Accept	Reject
4	70	1	62,8%	805	Accept	Reject
5	70	1	78,7%	833	Accept	Reject
6	70	1	83,8%	887	Accept	Reject
7	70	2	97,1%	937	Drop	Accept
8	70	2	90,8%	929	Drop	Accept
9	70	2	83,2%	904	Drop	Accept
10	70	2	75,5%	875	Drop	Accept

Berdasarkan Tabel 4 pada pengujian ke 7 terjadi peningkatan CPU yang melebihi dari 90%. Pengujian dilakukan dengan waktu yang berurutan agar dapat dilihat kenaikan memori yang dihasilkan. Kemudian setelah dilakukan *reject* koneksi pada pengujian ke 8, terlihat penurunan penggunaan CPU dan memori tiap detiknya.

4.3 Pengujian Skenario *Request* Tanpa *Scaling*

Pengujian skenario *request* tanpa *scaling* dilakukan dengan tujuan untuk mengetahui kinerja *storage* server saat dilakukan *request* tanpa melakukan *reject* koneksi. Hasil yang didapatkan akan dilihat perbandingannya dengan server yang dilakukan *scaling*. Pada tahap ini akan dilakukan pengujian sebanyak 20 pengujian agar data yang didapat lebih akurat.

Pada Tabel 5 dapat dilihat pengujian 100 *request* tanpa *scaling* terjadi peningkatan kinerja CPU dan memori. Pengujian dilakukan dengan waktu yang berurutan agar dapat dilihat kenaikan memori yang dihasilkan. Selanjutnya dihitung nilai kenaikan rata-rata CPU, memori dan *response time* yang merujuk pada persamaan (2.1). Kenaikan rata- rata CPU adalah 33,10% perdetik. Sedangkan kenaikan rata-rata memori adalah 36,3 MB perdetik dan pada *response time* adalah 564,7 ms. Berikut dapat dilihat hasil pengujian *request* tanpa *scaling* dengan 100 *request* pada Tabel 5.

Tabel 5 Pengujian 100 *Request* Tanpa *Scaling*

No.	*Request*/detik	CPU (%)	Memori (Mb)	*Response time* (ms)
1	100	0,7%	275	563
2	100	33,6%	282	563
3	100	93,4%	291	564
4	100	100%	332	564
5	100	100%	385	566
6	100	100%	418	566
7	100	100%	470	564
8	100	100%	502	564
9	100	100%	558	565
10	100	100%	639	565
11	100	100%	687	565
12	100	100%	746	565
13	100	100%	790	564
14	100	100%	833	565
15	100	100%	889	565
16	100	100%	913	565
17	100	100%	944	565
18	100	100%	977	565
19	100	100%	979	566
20	100	**100%**	**979**	566

4.4 Pengujian Skenario *Request* Dengan *Scaling*

Pengujian 100 *request* dengan *scaling* dilakukan dengan tujuan untuk mengetahui apakah sistem yang dibuat dapat mengatasi peningkatan *storage*. Pada pengujian *iptables* digunakan untuk melakukan *reject* paket koneksi jika kinerja CPU telah melebihi 90%..

Pada Tabel 6 dapat dilihat bahwa telah terjadi penurunan penggunaan CPU dan memori pada pengujian ke 11. Pengujian ke 11 tidak terjadi *response time* karena koneksi telah ditolak ke server 1 oleh *iptables* sehingga selanjutnya tidak akan terjadi *request* pada server ataupun semua *request* ditolak oleh server. Selanjutnya dihitung nilai kenaikan rata-rata CPU, memori dan *response time* yang merujuk pada persamaan (2.1). Rata-rata penurunan CPU yang didapat adalah 6,07% perdetik dan penurunan pada memori adalah 2,9 MB perdetik. Hasil pengujian dengan *scaling* dapat dilihat pada Tabel 6. Pengujian dilakukan dengan waktu yang berurutan agar dapat dilihat kenaikan memori yang dihasilkan.

Tabel 6 Pengujian 100 *Request* Dengan *Scaling*

No.	*Request*/detik	CPU (%)	Memori (Mb)	*Response time* (ms)
1	100	0,7%	275	563
2	100	58,2%	282	563
3	100	95,9%	291	564
4	100	100%	332	564
5	100	100%	385	566
6	100	100%	418	566
7	100	100%	470	564
8	100	100%	502	564
9	100	100%	558	565
10	100	**100%**	**639**	565
11	100	96,1%	638	tidak ada
12	100	89,3%	636	tidak ada
13	100	80,2%	632	tidak ada
14	100	72,7%	628	tidak ada
15	100	70,3%	623	tidak ada
16	100	62,5%	619	tidak ada
17	100	60,1%	618	tidak ada
18	100	51,3%	613	tidak ada
19	100	48,3%	612	tidak ada
20	100	39,9%	610	tidak ada

4.5 Pembahasan

Dari hasil observasi yang telah dilakukan dapat dilihat kinerja *storage* pada server saat menerima *request* dengan dialihkan koneksi dan tanpa dialihkan koneksi. Terdapat beberapa poin pada pembahasan yang dilakukan pada penelitian

Pada pengujian dicari nilai selisih dan rata-rata CPU, memori dan *response time*. Cara menghitung nilai selisih adalah hasil pengujian dikurangi dengan hasil pengujian sebelumnya. Setelah didapatkan nilai selisih pada masing-masing pengujian maka selanjutnya akan dihitung nilai rata-rata dari nilai selisih yang didapat. Nilai rata-rata didapat dari jumlah total keseluruhan nilai selisih dibagi dengan banyaknya pengujian. Keseluruhan tentang cara menghitung nilai rata-rata adalah merujuk pada persamaan (2.1).

Pada pengujian keberhasilan monitoring status *storage* dibagi menjadi 3 penguian *sample* data *request* yaitu 25, 50 dan 100 *request*. Pada pengujian *sample* data 25 *request* data tertinggi CPU adalah 48,3% dan memori adalah 285 MB. Status yang

ditampilkan pada aplikasi adalah normal karena kenaikan kinerja CPU tidak melebihi 50%. Nilai selisih rata-rata peningkatan kinerja CPU adalah 11,22% perdetik dan nilai selisih rata-rata peningkatan kinerja memori adalah 2 MB perdetik. Kemudian pada *sample* data 50 *request* data tertinggi CPU adalah 73,0% dan data tertinggi memori 338 MB. Status yang ditampilkan pada aplikasi adalah *warning* atau *medium* karena kenaikan kinerja CPU telah melebihi 50%. Nilai selisih rata-rata peningkatan kinerja CPU adalah 14,46% perdetik dan nilai selisih rata-rata peningkatan memori adalah 12,6 MB. Selanjutnya pada *sample* data 100 *request* data tertinggi CPU adalah 100% dan memori adalah 470 MB. Status yang ditampilkan pada aplikasi adalah *critical* dikarenakan kinerja CPU telah melebihi dari 90%. Nilai selisih rata-rata peningkatan kinerja CPU adalah 33,1% perdetik dan nilai selisih rata-rata peningkatan memori adalah 39 MB perdetik.

Pada pengujian keberhasilan monitoring terdapat beberapa masalah seperti *delay* saat menampilkan data *storage* ke aplikasi monitoring. *Delay* disebabkan oleh memori server yang kecil. Aplikasi telah memakai setengah dari memori server saat dijalankan sehingga data tidak dapat ditampilkan secara *realtime*. Aplikasi dapat menampilkan status server namun dengan *range delay* selama satu menit.

Terdapat perbedaan nilai pada data yang ditampilkan aplikasi dengan data *storage* dari setiap pengujian *sample* data. Perbedaan nilai terjadi karena beberapa penyebab antara lain seperti data yang ditampilkan secara *realtime* memerlukan memori yang besar. Penyebab utama data tidak sama pada saat ditampilkan ke aplikasi monitoring adalah keterbatasan hak akses pada server. Data pada server *elastic compute* (EC2) tidak dapat dilakukan konfigurasi untuk mengambil data tersebut.

Pada pengujian keberhasilan *iptables* mengalihkan koneksi, *request* akan dialihkan ketika kinerja CPU melebihi 90%. Nilai kenaikan kinerja CPU tertinggi terdapat pada pengujian ke 7, yaitu 97,1% dengan kinerja memori sebesar 937 MB. Pada pengujian ke 7, *iptables* melakukan *reject* paket untuk menolak semua paket *request* yang masuk ke server 1. *Reject* koneksi *request* yang dilakukan pada server 1 menyebabkan kinerja CPU menurun dengan selisih penurunan sebesar 7,2% perdetik dan kinerja memori menurun dengan selisih penurunan sebesar 20,6 MB perdetik. Terdapat perbedaan waktu saat pengujian melakukan *reject* dan *accept* paket pada server 1. Paket akan terus diterima oleh server 1 tiap satu detik saat *iptables* melakukan *accept* koneksi. Selanjutnya ketika koneksi akan di-*reject* memiliki *delay* waktu sekitar 2 detik saat mengeksekusi koneksi *request*.

Pada pengujian skenario tanpa *scaling* terbagi menjadi 3 jenis pengujian *request* yaitu 25, 50 dan 100 *request* perdetik dengan masing-masing *request* dilakukan pengujian sebanyak 20 kali. Pada pengujian 25 *request* didapatkan nilai kenaikan untuk rata-rata penggunaan CPU sebesar 10,90% perdetik dan memori sebesar 3,1 MB perdetik dengan rata-rata *response time* adalah 290,6 ms. Pada pengujian 50 *request* didapatkan nilai kenaikan rata-rata penggunaan CPU sebesar 9,27% perdetik dan memori sebesar 10,5 MB perdetik dengan rata-rata *response time* adalah 291,1 ms. Selanjutnya pada pengujian 100 *request* didapatkan nilai kenaikan rata-rata CPU sebesar 33,10% perdetik dan memori sebesar 36,3 MB dengan rata-rata *response time* adalah 564,7 ms.

Pada pengujian skenario dengan *scaling*, pengujian dilakukan dengan 100 *request* perdetik. Pada tahap pengujian ini, akan bagi menjadi dua bagian. Pengujian ke 1 sampai 10 adalah pengujian *accept* koneksi 100 *request*, sedangkan pengujian 11 sampai 20 adalah pengujian koneksi yang di *reject* oleh *iptables* pada server. Selanjutnya didapatkan nilai rata-rata peningkatan CPU pada pengujian ke 1 sampai 10 sebesar 33,10% perdetik dan memori sebesar 42,4 MB dengan rata-rata *response time* sebesar 564,4 ms. Sedangkan pada pengujian 11 sampai 20 didapatkan nilai penurunan CPU sebesar 6,07% perdetik dan memori sebesar 2,9 MB perdetik. Sedangkan pada pengujian 11 sampai 20 tidak memilik nilai *response time* karena koneksi telah ditolak oleh server.

5 KESIMPULAN

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, didapatkan kesimpulan sebagai berikut:

1. Pada penelitian ini telah berhasil mengatasi peningkatan CPU dan memori menggunakan teknik scaling dengan cara melakukan reject atau penolakan koneksi dari protokol ICMP. *Tools* yang digunakan untuk melakukan reject koneksi adalah *iptables*..
2. Berdasarkan pengujian keberhasilan monitoring status storage, server akan menampilkan status critical jika kenaikan CPU pada server telah melebihi dari 90%.
3. Berdasarkan pengujian skenario *request* dengan *scaling*, dilakukan pengujian *reject* koneksi *request* untuk melihat penurunan kinerja dari CPU dan memori. Pada pengujian tersebut menghasilkan rata-rata penurunan kinerja CPU sebesar 6,07% perdetik dan memori sebesar 2,9 MB perdetik. Nilai rata-rata pada *response time* sebelum dilakukan *scaling* adalah 564,4 ms. *Reject* koneksi dilakukan dengan menekan *button active* pada aplikasi. Hasil yang diperoleh menunjukkan bahwa aplikasi telah berhasil menangani peningkatan *storage* pada server

6 SARAN

Saran untuk penelitian berikutnya adalah sebagai berikut:

1. Penelitian selanjutnya diharapkan peneliti menambah parameter dalam implementasi scaling agar sistem yang dijalankan memperoleh hasil yang lebih akurat...
2. Implementasi pada *scaling* server selanjutnya diharapkan dapat berjalan secara otomatis atau *auto* dengan konsep horizontal *scaling* agar mempermudah administrator dalam memonitoring server.

DAFTAR PUSTAKA

[1] S. Bhowmik, *Cloud Computing*, India: Cambridge University Press, 2017.

[2] F.L. Ferraris, D. Francheschelli and M.P Gioiosa, Evaluating the Auto Scaling Performance of Flexiscale and Amazon EC2 Clouds, Milano: Politecnico, 2012.

[3] C. Prihantoro, Komponen Dasar Implementasi Cloud Computing Strategy Kategori Software As A Service (SAAS) Dan Infrastructure As A Service (IAAS) Pada Instansi Perguruan Tinggi *Seminar Nasional Teknologi Informasi Dan Aplikasinya*, 2015, .

[4] M. Subramanian, *Network Management Principles and Practic*, *International Journal of Advanced Research in Computer and Communication Engineering*, 2016.

[5] Sharmila and Ramadevi, Analysis of Performance Testing on Web, Jakarta: Alex Media Komputindo, 2014.

[6] M.S. Adnan, Analisis Kinerja Web Server Dengan Metode Load Balancing Pada HAProxy, Yogyakarta: STIMIK AKAKOM Yogyakarta, 2017.

[7] Ferry, *Response Time Testing*, Binus University School of Information System, 2020.

[8] I. Hamida and A. P. Sujana, Analisis VPS Cloud Pada Database Server, 2017.

[9] H. Jusuf, Penggunaan Secure Shell (SSH) Sebagai Sistem Komunikasi Aman Pada Web Ujian Online, Bina Insani ICT Journal, 2015.

[10] E. Sediono, Perancangan Bandwidth Adaptif Dengan Memanfaatkan Incoming Internet Control Message Protocol (ICMP) Packet Request, 2012.

[11] Gregor and Purdy, Firewalls, NAT & Accounting (LINUX IPTABLE), United State of America: O'Reilly Media, Inc, 2004.

[12] Rasian and Musranto, Perbandingan Kinerja Pendekatan Virtualisasi, Jurnal Ilmu Komputer, 2009.

[13] A. Sandi, Alasan Mengapa Kamu Harus Menggunakan Framework Laravel, codepolitan.com, 2017.

[14] Harinaldi, Prinsip-Prinsip Statistik Untuk Teknik dan Sains, Jakarta: Erlangga, 2005.