ISSN. 1412-0100 VOL 17, NO 1, APRIL 2016

Penyeimbangan Beban pada Sistem Berorientasi Service di Lingkungan Cloud dengan menggunakan Algoritma Genetika dan Graf

Roswan Latuconsina
Gedung Barung Ruang TE1.02.03 (N203), Jl. Telekomunikasi No. 1, Bandung, 40257
Program Studi Sistem Komputer, Fakultas Teknik Elektro, Universitas Telkom
roswan@telkomuniversity.ac.id, roswan78@gmail.com

Abstrak

Aplikasi-aplikasi enterprise yang dibangun dengan konsep Service Oriented Architecture (SOA) memiliki perbedaan dengan aplikasi-aplikasi native yang telah berkembang sebelumnya. SOA adalah model arsitektur yang fleksibel dan interoperable. Pembangunan aplikasi yang berbasis pada service adalah sangat cost effective, disamping memungkinkan untuk me-reuse service-service yang dihasilkan. Cloud computing merupakan suatu teknologi yang memanfaatkan internet sebagai resource untuk komputasi yang dapat di-request oleh pengguna dan merupakan sebuah layanan dengan pusat server bersifat virtual atau berada dalam cloud itu sendiri. SOA dengan dukungan teknologi cloud memberikan banyak keuntungan. Banyaknya jumlah pengguna yang me-request service yang berlokasi di cloud menghantarkan masalah klasik, yaitu pengalokasian request (workload) pada node tertentu untuk mencapai utilitas sistem yang tinggi. Penelitian ini berupaya menggabungkan konsep graf dengan algoritma genetika sebagai mekanisme load balancing pada sistem SOA di lingkungan Cloud.

Kata kunci— *Service-Oriented Architecture (SOA), Cloud Computing, Load Balancing*

Abstract

Enterprises applications that built on the concept of Service Oriented Architecture (SOA) have differences with native applications that have been developed previously. SOA is an architectural model that offers flexible and interoperable solution for enterprises. Service-based applications are very cost effective as well as enable to reuse available services. Cloud computing is a new technology that utilizes internet as a resource for computing that can be requests by the consumer on virtualization manner. SOA with cloud provides many advantages. A large number of users who request the service that located in the cloud addressing a classic problem such as workload allocation problem on a particular node to achieve high utility. This paper discusses the concept of load balancing on SOA systems in Cloud environment. Various load balancing strategies in distributed system will be presented in this paper. We also propose a mechanism to overcome load imbalance. This mechanism combines graph techniques and genetics algorithm as a load balancing method on SOA system in cloud environment

Keywords— *Service-Oriented Architecture (SOA), Cloud Computing, Load Balancing*

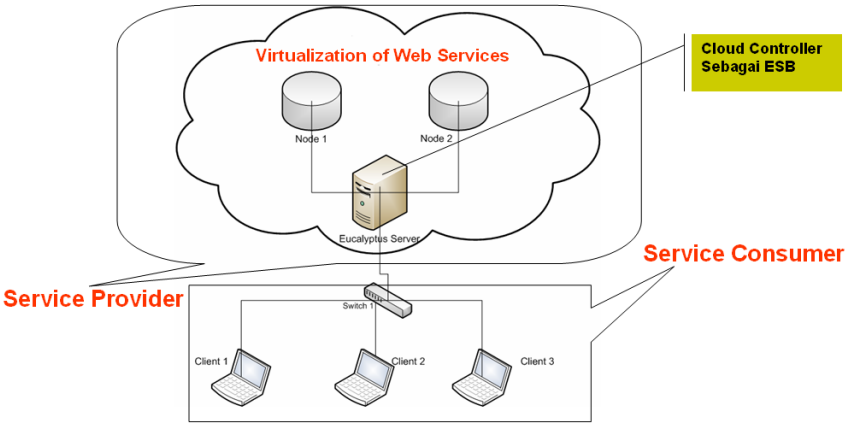
1. PENDAHULUAN

Service Oriented Architecture (SOA) merupakan paradigma baru yang inovatif dan saat ini terus dikembangkan. SOA adalah sebuah gaya arsitektur yang mendefinisikan suatu model interaksi antara tiga unit fungsional utama, yaitu *service provider*, *consumer* dan *registry*. *Service consumer* (*requestor*) berinteraksi dengan *service provider* untuk menemukan layanan (*service*) yang sesuai dengan kebutuhannya melalui pencarian di *registry*.

Sistem berbasis SOA yang dibangun akan mampu menangani perubahan-perubahan yang terjadi atau bersifat lebih fleksibel terhadap perubahan yang terjadi di logik aplikasi maupun di logik bisnis. Sebagai contoh, perubahan di logik aplikasi menyebabkan *service consumer* tidak dapat menemukan

service yang diinginkan, disebabkan karena *service provider* mengubah lokasi suatu *service* tanpa melakukan *updating* di *registry*.

Service-Oriented Architecture (SOA) dengan dukungan cloud muncul sebagai solusi yang kuat, yang memungkinkan interoperabilitas antar komponen perangkat lunak yang tersebar, atau lebih dikenal dengan Web Services (WS).



Gambar 1. Ilustrasi Sistem SOA di Lingkungan Cloud

Sisi Potensial Cloud Computing memberi keuntungan antara lain: (1) *Benefit* bagi para pelaku bisnis, yaitu minimalisasi biaya investasi infratruktur publik sehingga bisnis bisa lebih terfokus pada aspek fungsionalistasnya, (2) Bagi *application developer*, layanan Platform-as-a-Service (PaaS) memungkinkan pengembangan dan implementasi aplikasi dengan cepat sehingga meningkatkan produktivitas, (3) Bagi para praktisi yang bergerak di industri IT, hal ini berarti terbukanya pasar baru bagi industri jasa pengembangan teknologi informasi, (4) Bagi pebisnis di bidang infrastruktur, hal ini merupakan peluang yang besar karena dengan meningkatnya penggunaan layanan Software-as-a-Service (SaaS) ini akan meningkatkan penggunaaan bandwidth internet, (5) Integrasi aplikasi dengan berbagai device.

David Linthicum [1] menjelaskan hubungan antara cloud computing dan SOA adalah bahwa cloud computing menyediakan sumber daya TI yang dapat dimanfaatkan sesuai kebutuhan (*on demand*), termasuk sumber daya untuk host data, service, dan proses. Dengan demikian, perusahaan memiliki kemampuan untuk memperluas SOA di luar *firewall* perusahaan bagi provider cloud.

Ben Letaifa, dkk [2] mengatakan bahwa SOA menyediakan *loose coupling* antar aplikasi, sementara cloud computing memberikan *loose coupling* antara aplikasi dan hardware.

Penggunaan infrastruktur cloud pada sistem berorientasi service (SOA) memberikan banyak keuntungan di satu sisi. Namun disisi lain dihadapkan pula dengan permasalahan kinerja sistem. Misalnya saja penurunan utilitas sistem, masih tingginya response time yang diberikan sistem, dan permasalahan heterogenitas pada infrastruktur cloud. Permasalahan lain yang menjadi tantangan dan *state-of-the-art* dari penelitian pada cloud adalah tentang *live migration* dan virtualisasi. Jika dikaji lebih mendalam tentang semua permasalahan tersebut dan diambil irisannya, maka semuanya dapat bermuara pada satu permasalahan, yaitu persoalan *Load Balancing*.

Strategi Load balancing ini kemudian timbul sebagai solusi atas adanya kondisi ketidakseimbangan beban (*imbalance*), *bottleneck* maupun *overload* pada sistem. Sebuah sistem dikatakan overload jika perbandingan beban dan kapasitas sistem itu > 1, atau dengan kata lain utilitas sistem (u) > 1.

Beberapa penelitian tentang *load balancing* pada cloud telah dilakukan oleh Jinhua [3], Yi Zhao [4] dan Wei-Tek Tsai [5]. Ketiganya mengembangkan algoritma penjadwalan yang berbeda untuk diterapkan pada sisi virtual machine. Namun demikian, ketiganya masih sangat bergantung pada jenis virtual machine (VM) tertentu yang digunakan. Penerapan strategi load balancing pada VM yang berbeda tidak menjamin perbaikan kinerja pada sistem tersebut.

Aimrudee [6] mengembangkan mekanisme load balancing yang berbeda untuk sistem yang berbasis SOA, yaitu dengan menggunakan Enterprise Service Bus (ESB) sebagai router yang menentukan pe-*rute*-an permintaan client ke server yang akan melayani permintaan itu. Aimrudee mengakategorikan service-service yang ada, dan menermpatkan service-service yang sejenis ke dalam sebuah *server group* yang sama. Dalam skala yang lebih luas, mekanisme ini belum menjanjikan utilitas sistem yang baik.

Untuk menjawab tantangan dan mengatasi kelemahan-kelemahan dari penelitian-penelitian sebelumnya, maka dilakukanlah penelitian tentang mekanisme load balancing yang memadukan karakteristik SOA dan Cloud Computing sehingga kinerja sistem dapat menjadi lebih baik dengan mempertimbangkan segala keterbatasan penelitian-penelitian sebelumnya.

2. STUDI LITERATUR

Persoalan load balancing sudah menjadi permasalahan klasik dalam sistem terdistribusi. Berbagai strategi load balancing telah banyak dikemukakan oleh para ilmuwan yang berasal dari lingkungan akademik maupun industri. Secara garis besar, beragam strategi itu hanya dikelompokkan menjadi dua jenis saja, yaitu *static load balancing* dan *dynamic load balancing.*

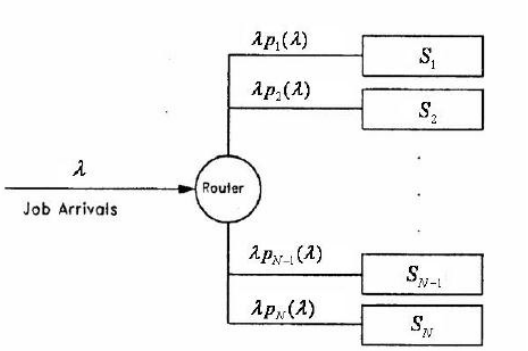
Proses *load balancing* sebenarnya merupakan proses fleksibel yang dapat diciptakan dengan berbagai cara dan metode. Proses ini dapat dilakukan oleh sebuah perangkat tertentu atau sebuah software khusus saja. Suatu algoritma *load balancing* sebaiknya berupaya untuk mencapai tujuan berikut :

- Meminimasi ketidakseimbangan beban (*load imbalance*). Menyediakan Quality of Service (QoS) yang terbaik, dimana setiap node memiliki utilisasi yang sama.
- Meminimasi jumlah beban yang berpindah. Memindahkan sejumlah besar beban dengan menggunakan *bandwidth* dan mungkin tidak *feasible* jika sebuah beban node berubah secara cepat sehubungan dengan waktu yang dibutuhkan untuk memindahkan obyek tersebut.

Jan Kuting [7], mengajukan sebuah pendekatan load balancing untuk sistem berorientasi service yang berbasis pada response time. Pendekatan ini didasarkan pada perhitungan rasio antara *response-time* t_b saluran terbaik dan response-time saluran terukur t_i. Bobot normal per saluran (indeks kinerja) dapat didefinisikan sebagai $f_i = \frac{t_b}{t_i}$. Diasumsikan bahwa $t_b \leq t_i$ karena t_b telah dipilih sebagai yang terbaik (*response-time* terendah). Kuting membandingkan pendekatan berbasis response-time ini dengan tiga pendekatan laiinya, yaitu *Queue-Length-Based approach, Shortest Queue* dan Round Robin. Kuting juga memperkenalkan konsep *Virtual Service* yang terdiri atas beberapa *service instances* (saluran).

Leonidas Georgiadis [8] mengajukan *min-max policy* (MMP) untuk meminimasi maksimum rerata response-time job pada server tertentu dalam sistem yang heterogen. Diasumsikan kedatangan job terjadi menurut distribusi poisson dengan tingkat kedatangan λ jobs/second. Router mengirimkan job ke server S_i dengan probabilitas $p_i(\lambda)$. Response time sebuah job di server S_i didefinisikan sebagai waktu yang dibutuhkan sejak job tiba di S_i hingga job tersebut selesai diproses dan meninggalkan sistem. Fungsi response-time $R_i(x)$ yang merupakan rerata response-time sebuah job pada S_i dengan tingkat kedatangan x pada sistem. Jika $p_i(\lambda)$ dan $R_i(x)$ diberikan, maka rata-rata respose-time sebuah job yang datang ke router dapat dihitung sebagai berikut.

$$R(\lambda) = \sum_{i=1}^{N} R_i(\lambda p_i(\lambda)) p_i(\lambda)$$



Gambar 2. Sistem multi-komputer dengan *central job router* [8]

Hal yang sama juga telah dilakukan oleh Valeriy Naumov [9]. Ia menggunakan sebuah fungsi biaya $c_i(x)$ untuk mengalokasikan job ke server yang sesuai.

$$c_i(x) = \frac{d(xR_i(x))}{dx}$$

Naumov juga melakukan perbaikan terhadap metode minimum response-time atau *Shortest Expected Delay* (SED). Dalam SED, sebuah job atau task dijadwalkan pada server dengan response-time terkecil yang diharapkan, sehingga

$$\text{Minimasi} \quad = \frac{(s_i + 1)}{\mu_i}$$

dengan s_i adalah jumlah task (job) pada server *i* dan μ_i adalah taraf pemrosesan job. Dengan kata lain, metode SED menyetarakan response-time pada server aktif, yang mengarah pada minimalisasi response-time maksimum. Modifikasi yang dilakukan **Naumov** adalah dengan menyetarakan response-time dikalikan dengan akar kuadrat dari taraf pemrosesan job, sehingga ekspresinya menjadi

$$\text{Minimasi} \quad = \frac{(s_i + 1)}{\sqrt{\mu_i}}$$

Y F Hu dan R J Blake [10] memperkenalkan sebuah metode load balancing dinamik yang optimal dengan menggunakan konsep Graf. Misal sejumlah *p* prosesor dan (*V, E*) adalah graf prosesor, dimana V=(1,2, ...,p) merupakan sekumpulan simpul yang merepresentasikan sebuah prosesor, dan *E* adalah sekumpulan sisi. Dua buah simpul *i* dan *j* membentuk sebuah sisi jika prosesor *i* dan *j* membagi sebuah *boundary of partitioning*. Setiap prosesor *i* adalah sebuah skalar l_i yang menggambarkan beban atau *load* pada prosesor. Rerata *load* setiap prosesor adalah

$$\vec{l} = \frac{\sum_{i=1}^{p} l_i}{p}$$

Setiap sisi (i,j) juga memiliki sebuah skalar δ_{ij} yang merepresentasikan jumlah load yang akan dikirim dari prosesor i ke prosesor j. Variabel δ_{ij} adalah direksional, sehingga

$$\delta_{ij} = - \delta_{ij} \qquad (1)$$

Hal ini berarti jika prosesor i mengirim sejumlah δ_{ij} ke prosesor j, maka prosesor j menerima sejumlah yang sama (mengirim $-\delta_{ij}$).

Penjadwalan *load balancing* akan membuat beban setiap prosesor sama dengan rerata beban, sehingga

$$\sum_{\{j|(i,j)\in E\}} \delta_{ij} = l_i - \vec{l}_j, i = 1,2,..., p \qquad (2)$$

Jika i > j dan (i,j) $\in E$, simpul i akan bertindak sebagai kepala sisi (i,j) dan j sebagai ekor. Karena persamaan (1), maka δ_{ij} akan menjadi variabel jika i adalah kepala sisi (i,j), dan $-\delta_{ij}$ menjadi variabelnya jika i adalah ekor.
Jumlah persamaan tidak melebihi jumlah simpul minus satu. Jumlah variabel pada sistem dari persamaan (2) adalah sama dengan jumlah sisi pada graf. Biasanya terdapat lebih banyak sisi dibanding simpul dalam sebuah graf. Dalam setiap kasus graf $|E| \geq |V| - 1$, dimana |E| dan |V| adalah jumlah sisi dan simpul pada graf.
Dari persamaan (2) diinginkan sebuah solusi yang meminimasi pergerakan data. Misalkan A adalah matriks yang berhubungan dengan (2), x adalah vektor δ_{ij} dan b adalah variabel di bagian kanan persamaan. Diasumsikan aturan Euclidean dari pergerakan data digunakan sebagai metrik, sehingga biaya komunikasi antara dua prosesor adalah sama. Persamaannya menjadi :

$$\text{Minimasi} = \tfrac{1}{2}\, x^T x,$$
$$\text{dimana } Ax = b, \tag{3}$$

A adalah matriks |V| × |E| , dimana

$$(A)_{ik} = \begin{cases} 1, & \text{jika vertex i adalah kepala sisi k,} \\ -1, & \text{jika vertex i adalah ekor sisi k,} \\ 0, & \text{jika bukan keduanya} \end{cases}$$

Terapkan kondisi tersebut diatas untuk optimasi pada persamaan (3).

$$x = A^T \lambda \tag{4}$$

Dimana λ adalah vektor pengali Lagrange. Substitusi ke (2) sehingga

$$L\lambda = b \tag{5}$$

dengan $L = A\,A^T$ adalah matriks ukuran |V| × |V|.

Permasalahan menentukan strategi *load balancing* yang optimal membuahkan sebuah persamaan linier seperti pada persamaan (5). Matriks *L* pada dasarnya merupakan matriks Laplacian dari sebuah graf berdimensi |V| × |V|., yang didefinisikan sebagai berikut.

$$(L)_{ij} = \begin{cases} -1, & \text{jika } i \neq j \text{ dan sisi } (i,j) \in E, \\ deg(i), & \text{jika } i = j, \\ 0, & \text{jika tidak keduanya} \end{cases} \tag{6}$$

Disini *deg(i)* adalah derajat simpul i pada graf.

Ketika vektor Lagrange λ ditemukan melalui persamaan (5), maka melalui persamaan (4) dan menurut bentuk khusus A^T (tiap baris matriks hanya mempunyai dua nilai *non-zero*, yakni 1 dan -1) dapat ditentukan jumlah beban atau *load* yang ditransfer dari prosesor i ke prosesor j adalah $\lambda_i - \lambda_j$,

3. METODE PENELITIAN

Tahapan penelitian yang dilakukan adalah sebagai berikut :

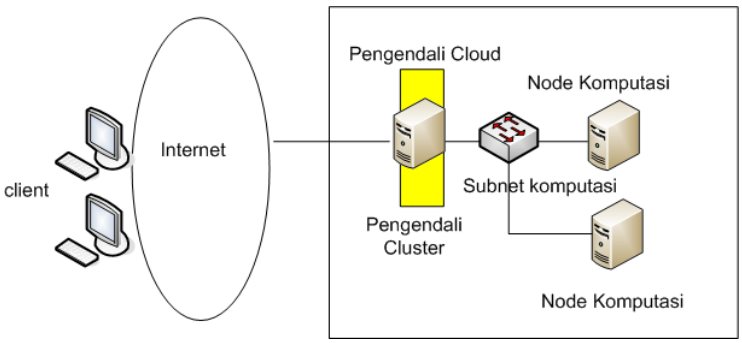
1. Preliminary Research. Identifikasi dan penilaian terhadap berbagai metodologi penelitian kinerja sistem. Melakukan riset paper mengenai beragam metode *load balancing* pada sistem terdistribusi.
2. Penyiapan kasus, Mempersiapkan sebuah sistem berbasis SOA dan penciptaan service-service yang memenuhi karakteristik SOA.
3. Penyiapan Infrastruktur. Membangun sebuah model infrastruktur cloud skala laboratorium. Salah satu yang menjadi model acuan adalah konfigurasi Eucalyptus (Gambar 1).
4. Perancangan Metode *Load Balancing*. Perancangan awal terhadap metode *load balancing* dilakukan berdasarkan hasil modifikasi dan komparasi metode *Load Balancing* yang *feasible*.

5. Pengujian metode *load balancing*. Dilakukan melalui sebuah simulasi dan eksperimen berskala laboratorium. Mengamati perilaku dalam berbagai skenario simulasi. Jika model cukup stabil, dilanjutkan ke eksperimen lab. Jika belum stabil, iterasi proses pengembangan secara menyeluruh sampai keadaan stabil tercapai.
6. Refleksi / pengembangan metode *load balancing*. Refleksi dapat dilakukan setelah simulasi atau setelah percobaan laboratorium dilakukan. Jika simulasi dan percobaan laboratorium telah dicapai, dilakukan validasi terhadap hasil. Melakukan identifikasi pengembangan metode serta upaya-upaya untuk melakukan pengembangan metode *load balancing*.
7. Penarikan kesimpulan terhadap hasil yang diperoleh.

4. PERANCANGAN SISTEM

4.1 Perancangan Infrastrukur Cloud

Untuk kebutuhan eksperimen dibangun sebuah infrastruktur cloud yang berbasis pada model Eucalyptus. Dalam percobaan, Cloud Controller (ClC) diset sebagai Cluster Controller (CC), dan Storage.

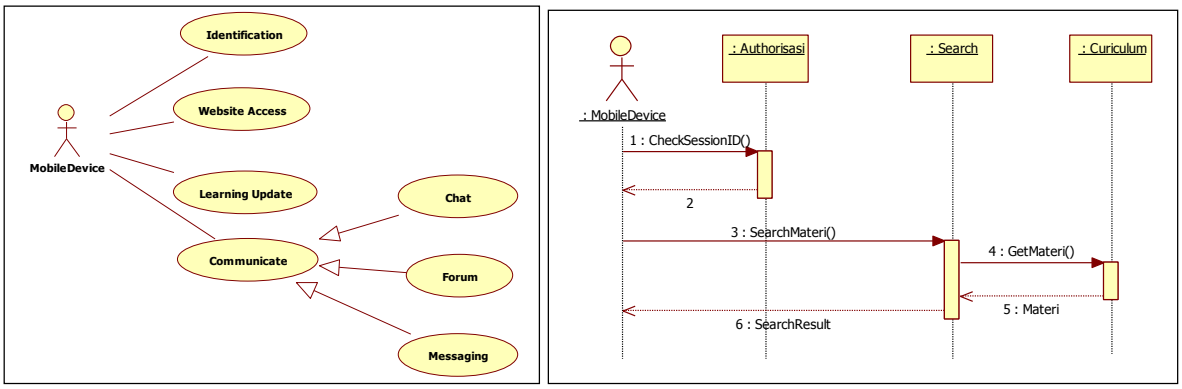


Gambar 3. Sistem cloud yang dibangun menggunakan arsitektur *Eucalyptus*

Node komputasi atau Node Controller (NC) merupakan server yang memiliki spesifikasi teknis virtualisasi dengan kemampuan komputasi. Node-node komputasi yang terhubung pada subnet membentuk cluster yang dikendalikan oleh pengendali cluster atau Cluster Controller. Pengendali storage merupakan tempat penyimpanan software atau image yang akan dieksekusi; Pengendali cloud merupakan server cloud yang beroperasi melayani client melalui web services.
3 (tiga) buah Node Controller (NC) disetup dengan spesifikasi teknis sebagai berikut :NC 1 : AMD NeoX2, 4 GB RAM; NC 2 : Xeon2Core, 4 GB RAM; dan NC 3 : Core2Quad, 4 GB RAM.
Untuk kebutuhan virtualisasi digunakan KVM (Kernel Virtual Machine). Tiap node komputasi terdapat virtual machine, yang merupakan wadah aplikasi. Aplikasi ini yang diakses oleh pengguna cloud. Dalam model ini, pengguna cloud tidak perlu menginstalasi aplikasi secara lengkap melainkan dapat secara fleksibel memilih fungsi aplikasi yang sesuai dengan kebutuhannya.

4.2 Perancangan Layanan Software as a Service (SaaS)

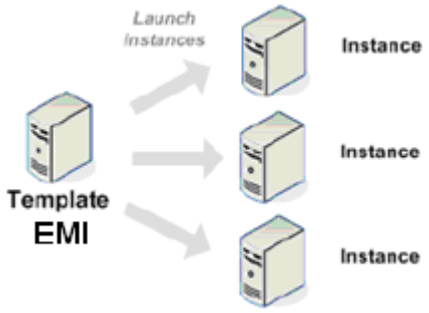
Untuk kebutuhan percobaan dibangun sebuah aplikasi (service) mobile learning yang memiliki beberapa layanan, yaitu wsIdentification, wsWebsiteAccess, wsLearningUpdate, dan wsCommunicate. Selanjutnya dibuat *Eucalyptus Machine Image* (EMI) yang berisi aplikasi, *library*, data dan pengaturan konfigurasi yang terkait, atau dapat juga dengan memilihnya langsung dari *library* EMI yang tersedia secara global.



(a) (b)

Gambar 4. (a) Use case diagram sistem mobile learning (b) Sequence diagram proses akses materi

Kemudian EMI diunggah ke *Eucalyptus* Dari EMI, pengguna me-*launch* instans, yang menjalankan copy dari EMI. Pengguna dapat meluncurkan beberapa instans dari EMI, seperti ditunjukkan pada gambar berikut. Instans pengguna tetap berjalan sampai dihentikan oleh pengguna sendiri, atau sampai tercapai kondisi instans gagal. Jika sebuah instans gagal, pengguna dapat me-*launch* instans yang baru dari EMI.



Gambar 5 . Proses me-*launch* instans pada Ubuntu Enterprise Cloud (UEC)

4.3 Perancangan mekanisme Penyeimbang Beban

Pada penelitian ini diajukan sebuah mekanisme untuk penyeimbangan beban kerja dengan mempertimbangkan biaya *load balancing* dan utilitas sistem Hal ini disebabkan karena dimungkinkannya suatu kondisi dimana *load balancing* tidak perlu dilakukan karena biaya yang timbul akan melampaui manfaat dari penyeimbangan beban itu sendiri. Untuk mengatasi ini adalah mungkin untuk mengkombinasikan beberapa strategi load balancing yang ada.

Secara umum, prosedur yang dibuat terdiri dari dua tahap utama disebabkan mekanisme penyeimbang beban yang terletak di Cloud Controller (CLC) dan Cluster Controller (CC). Tahap pertama menerapkan Algoritma Genetika untuk mencapai utilitas tinggi dari sistem cloud. Tahap kedua mengimplementasikan konsep Graph untuk mengoptimalkan fungsi biaya. Prosedur secara rinci sebagai berikut:

1. Client kirim permintaan (*request*) suatu *image*
2. Cloud Controller (ClC) melihat Table of Service (ToS)
3. ClC mengirim informasi ke Cluster Controller (CC) yang utilitas sistemnya terendah. Untuk melakukan hal ini, algoritma genetika diterapkan
4. CC memilih Node Controller (NC) dengan biaya optimal dengan menerapkan metode graf.
5. Akhirnya, NC terpilih akan membuat instans VM dari image yang diminta.

Beberapa asumsi dan batasan yang digunakan pada percobaan ini adalah :

- Kondisi node (NC) heterogen dengan beragam kapasitas (jumlah instans).

- Setiap *request* merupakan *job*
- Setiap *job* memiliki *weight* yang sama, dikarenakan sebuah *request* meminta sebuah instans (berada dalam VM); Sehingga 1 request = 1 instans
- Kebutuhan kapasitas setiap service dianggap sama
- Tingkat kedatangan *request* terjadi menurut distribusi poisson

4.3.1 Metode Graf

Metode graf dapat digunakan untuk menyelesaikan permasalahan optimasi yang memiliki fungsi tujuan dengan banyak variabel. Pada kasus ini, pertambahan Virtual Machine (VM) pada suatu Node Controller (NC) dan meningkatnya waktu layanan, menurut informasi historis dan kondisi saat ini, menjadikan beban setiap NC dapat diketahui. Beban NC ke-i pada periode T atau *L(i,T)* dihitung dengan formula :

$$L(i,T) = \sum_{j=1}^{m_i} \overline{V_i(j,T)} \qquad (7)$$

dengan *V* adalah VM pada NC, dan *m* adalah jumlah VM pada NC.
Dengan menggunakan persamaan (5) pada bagian terdahulu dan persamaan (7) di atas, kita dapat mengoptimalkan beban setiap NC dengan cara :
Langkah 1 : Tentukan rerata beban (*load*), dan variabel kanan pada persamaan (5)
Langkah 2 : Selesaikan $L\lambda = b$ untuk memperoleh λ;
Langkah 3 : Tentukan jumlah load yang akan ditransfer ($\lambda_i - \lambda_j$).

4.3.2 Algoritma Genetika

Algoritma Genetika adalah algoritma pencarian berdasarkan pada prinsip-prinsip evolusi dan genetika alami. Algoritma Genetika menggabungkan eksploitasi hasil masa lalu dengan eksplorasi daerah baru dari ruang pencarian. Dengan menggunakan teknik *survival of the fittest* yang dikombinasikan dengan pertukaran informasi terstruktur acak, algoritma ini dapat meniru beberapa bakat inovatif pencarian. Banyak peneliti telah menggunakan algoritma genetika untuk memecahkan masalah penjadwalan Algoritma ini bekerja dengan sebuah populasi yang terdiri dari individu-individu, yang masing-masing individu mempresentasikan sebuah solusi yang mungkin bagi persoalan yang ada. Dalam kaitan ini, individu dilambangkan dengan sebuah nilai fitness yang akan digunakan untuk mencari solusi terbaik dari persoalan yang ada. Berikut ini *pseudocode* dari algoritma genetika.

```
Input:
P: Populasi awal yang di-generate secara acak
Pc, Pm : nilai probabilitas crossover dan mutasi
MAXGEN : nilai maksimum generasi
N : Jumlah populasi |p|
Output:
X: Individu terbaik dari P
Method:
HiungFitness(P)
While (generasi < MAXGEN ^ (Tidak Konvergen) do
        M ← Reproduksi(P)
        O  ← Crossover(M, Pc)
        O ← Mutasi(O, Pm)
        HitungFitness(P)
        P ← Select(P,O)
        Generasi ← generasi + 1
End while
```

Gambar 6. Pseudocode Algoritma Genetika Sederhana

Fungsi tujuan adalah komponen yang paling penting dari metode optimasi algoritma genetika. Tujuan utama di sini adalah untuk meminimasi waktu tanggap *(response time)*, memaksimalkan utilitas NC dan penyeimbangan beban di semua NC. Kemudian, fungsi tujuan dimasukkan ke dalam fungsi fitness. Fungsi fitness ini kemudian akan digunakan untuk mengukur kinerja dalam kaitannya dengan tujuan algoritma.

Karena persoalan load balancing ini bertujuan untuk meminimalkan *response time*, maka fungsi fitness-nya adalah inversi dari total *response time* yang diperoleh. Sementara Utilitas NC merupakan pembagian *response time* dengan maxspan. Maxspan adalah *response time* terbesar yang ada di dalam sistem. Rata-rata utilitas adalah jumlah seluruh utilitas NC dibagi banyaknya NC dalam sistem.

Fungsi Fitness = (1/ Maxspan) * Rata-rata Utilitas

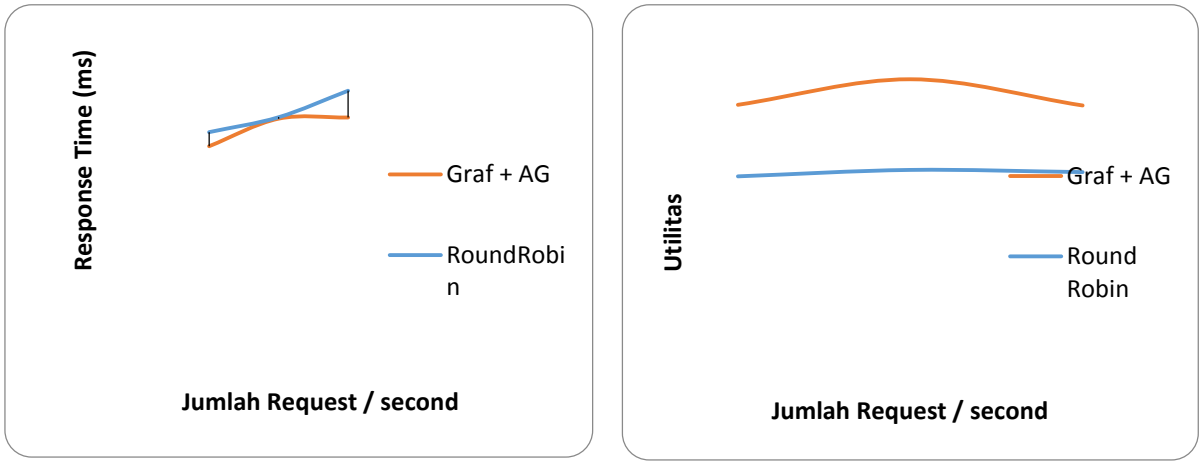
5. HASIL DAN PEMBAHASAN

Untuk menguji bahwa suatu web service dapat mendukung sebuah permintaan, maka beban tertentu harus dihasilkan. Dalam percobaan ini digunakan sebuah load generator LoadUI. LoadUI adalah sebuah solusi pengujian yang free dan open source serta lintas platform. Dengan antarmuka, visual drag-and-drop, memungkinkan pengguna untuk membuat, mengkonfigurasi dan mendistribusikan uji beban secara interaktif dan real-time. Dalam lingkungan pengujian, loadUI menyediakan cakupan uji yang lengkap dan mendukung semua protokol standar dan teknologi. Dalam eksperimen digunakan tiga skenarion beban yaitu beban 100 request/s, 1000 request/s, dan 10000 request/s. Pada tabel 1disajikan rekapitulasi hasil uji coba *request image web service mobile learning* dengan menggunakan mekanisme round robin (default *Eucalyptus*), serta mekanisme graf dan algoritma genetika (AG) sebagai mekanisme penyeimbang beban.

Tabel 1 Cuplikan data *Response Time* dan Utilitas

Load (req/s)	Round Robin		Graf + Algoritma Genetika (AG)	
	Response Time (ms)	Utilitas (%)	Response Time (ms)	Utilitas (%)
100	17710	64,11	16431	82,16
1000	19067	65,73	18937	88,61
10000	21478	65,18	19042	81,99

Response time dan utilitas sistem yang dihasilkan dengan menggunakan mekanisme penyeimbangan beban round robin (default dari eucalyptus) dan mekanisme kombinasi graf dan algoritma genetika ditampilkan pada grafik berikut.



(a) (b)

Gambar 7.Grafik (a) *Response Time* (b) Utilitas. Perbandingan Mekanisme Round Robin dengan Metode Graf dan Algoritma Genetika (AG)

Dari hasil percobaan diperoleh *response time* yang lebih kecil (lebih baik) dengan menggunakan metode graf + AG dibandingkan dengan mekanisme round robin untuk penyeimbangan beban. Grafik yang ditunjukkan pada Gambar 7a terlihat tidak begitu linier. Hal ini disebabkan salah satunya oleh kondisi trafik jaringan saat percobaan dilakukan yang tidak stabil. Untuk utilitas sistem, metode graf + AG menghasilkan utilitas yang jauh lebih baik dibanding round robin, yakni di atas 80%.

4. KESIMPULAN

Dalam tulisan ini, telah dibahas beberapa teknik *load balancing* di lingkungan konvensional maupun di lingkungan cloud. Disajikan pula beberapa konsep dan penelitian mengenai teknik optimasi terkait penyeimbangan beban kerja. Jika proses atau *task / job* menjadi target *load balancing* di lingkungan konvensional, maka di lingkungan cloud yang menjadi target adalah Virtual Machine (VM). Untuk menjalankan VM biasanya membutuhkan banyak waktu. Waktu tanggap (*response time)* dan utilitas Node Controller selalu menjadi parameter untuk menyeimbangkan beban kerja. Mengkombinasikan metode graf dan algoritma genetika untuk penyeimbangan beban di sebuah sistem berbasis SOA di lingkungan cloud adalah salah satu alternatif yang dapat digunakan serta cukup efektif untuk menghasilkan kinerja sistem yang lebih baik.

DAFTAR PUSTAKA

[1] David S. Linthicum, "*Cloud Computing and SOA convergence in your Enterprise*", Addison Wesley, 2010.

[2] Asma Ben Letaifa, Amel Haji, Maha Jebalia, Sami Tabbane, "*State of The Art and Research Challenges of New Services Architecture Technologies: Virtualization, SOA and Cloud Computing*", International Journal of Grid and Cloud Computing, December 2010.

[3] Jianhua Hu, Jianhua Gu, Guofei Sun, Tianhai Zhao, "*A Scheduling on Load Balancing of Virtual Machine Resources in Cloud Computing Environment*", IEEE 3rd International Symposium on Parallel Architectures, Algorithms and Programming, 2010.

[4] Yi Zhao, Wenlong Huang, "*Adaptive Distributed Load Balancing Algorithm based on Live Migration of Virtual Machines in Cloud*", IEEE 5th International Joint Conference on INC, IMS and IDC, 2009.

[5] Wei-Tek Tsai, Xin Sun, Qihong Shao, Guanqiu Qi, "*Two-Tier Multi-Tenancy Scaling and Load Balancing*", IEEE International Conference on E-Business Engineering, 2010.

[6] Aimrudee Jongtaveesataporn, Shingo Takada, "*Enhancing Enterprise Service Bus Capability for Load Balancing*", WSEAS Transaction on Computer Vol. 3, Maret 2010.

[7] Jan Kuting, Helmut Dispert, Joseph Morgan, "*Client-Based Adaptive Load Balancing in Service-Oriented Systems*", 2009.

[8] Leonidas Georgiadis, Christos Nikolaou, Alexander Thomasian, "*A fair Workload Allocation Policy for Heterogeneous*", Journal of Parallel Distributed Computing, 2004: 507-519

[9] Valeriy Naumov, "*Minimisation of the Average Response Time in a Cluster of servers*", International Conference on Computer System and Technologies, 2005.

[10] Y F Hu, R J Blake, "*An optimal dynamic load balancing Algorithm*", citeseer.ist.psu.edu/121199.html, 1995.