Load Balancing pada Cloud Computing dengan Sumber Daya Terbatas Menggunakan Penggabungan Algoritma ESCE dan Throttled

Endang Wahyu Pamungkas[1], Divi Galih Prasetyo Putri[2]
[1, 2)] Jurusan Teknik Informatika, Fakultas Teknologi Informasi,
Institut Teknologi Sepuluh Nopember,
Jalan Raya ITS, Kampus ITS Sukolilo, Surabaya, 60111, Indonesia
e-mail: endang.wahyu.pamungkas10@mhs.if.its.ac.id[1)], divi.galih10@mhs.if.its.ac.id[2)]

Diterima 13 Mei 2015
Disetujui 10 Juni 2015

***Abstract*— Recently cloud computing technology has been implemented by many companies. This technology requires a really high reliability that closely related to hardware specification and management resource quality used. Adequate hardware would make resource allocation easier. On the other hand, resource allocation will be harder if the resources are limited. This is a common condition in a developing cloud service provider. In this paper, a load balancing algorithm to allocate resources in cloud computing environment that has limited resources has been proposed. This algorithm is developed by taking the advantages of the existing algorithms, Equally Spread Current Execution and Throttled. We merge those algorithm without losing the advantages and we try to eliminate the shortcoming of each algorithm. The result shows that this algorithm is able to give a significant improvement in the limited resources environment. In addition, the algorithm also able to compete with the other algorithm in the more adequate resource environment. Based on the consistent results, this algorithm is expected to be more adaptive in different resources environment.**

Index Terms—*cloud computing, load balancing, throttled algorithm, equally spread current execution algorithm.*

I. PENDAHULUAN

Dalam beberapa tahun terakhir *cloud computing* telah menyita perhatian praktisi di dunia industri. *Cloud computing* tidak lagi hanya menjadi topik di dalam riset, tetapi sudah menjadi paradigma dalam dunia bisnis. Secara umum *cloud computing* dibagi menjadi tiga yaitu Infrastructure as a Service (IaaS), Software as a Service (SaaS), dan juga Platform as a Service (PaaS) [1] [2]. Dengan *cloud computing*, pelanggan tidak perlu membeli aplikasi yang dibutuhkan, melainkan hanya perlu menyewa aplikasi tersebut secara *online* dengan sistem pembayaran *pay-per-use* [1]. Keuntungan utama dari pemanfaatan *cloud computing* adalah sisi ekonomis serta efisiensi. Sisi ekonomis didapatkan oleh pihak pelanggan layanan yang hanya perlu menyewa IT *infrastructure*, perangkat lunak, maupun servis dalam jangka waktu tertentu sesuai dengan kebutuhan [3] [4]. Sehingga memudahkan perusahaan baru yang belum mampu membeli lisensi. Kemudian efisiensi didapatkan dari mobilitas, karena semua kegiatan IT dapat dilakukan hanya menggunakan layanan internet [3] [5] [6].

Saat ini *cloud computing* sudah berkembang dengan sangat pesat sehingga meningkatkan jumlah pengguna yang mengakses *cloud service*. Salah satu isu yang muncul akibat semakin meningkatnya pengguna ini adalah masalah *load balancing* [7]. Penyedia layanan harus mampu menjamin reliabilitas dari layanan baik disaat yang sibuk maupun normal [8]. Naik turunnya jumlah pengguna yang mengakses layanan ini harus diantisipasi dengan melakukan manajemen alokasi sumber daya dengan baik. Dengan sistem load balancing bagus akan menguntungkan bagi pelanggan yang mendapat layanan dengan *responsibility* tinggi. Selain itu Cloud Service Provider (CSP) yang memiliki algoritma *load balancing* yang baik akan semakin dipercaya oleh pelanggan serta dapat mengefisiensikan penggunaan sumber daya secara optimal. Permasalahan *load balancing* atau lebih popular dengan nama *job scheduling* ini merupakan permasalahan NP-Complete [9].

Secara umum algoritma *load balancing* diklasifikasikan menjadi dua yaitu *static* dan juga *dynamic*. Algoritma *static* cocok untuk lingkungan yang stabil dan juga homogen, tetapi peformanya akan turun jika atributnya berubah secara dinamis. Sedangkan algoritma *dynamic* lebih bersifat fleksibel terhadap perubahan atribut yang mungkin terjadi [10]. Permasalahan *load balancing* di lingkungan *cloud computing* lebih cocok dengan algoritma *dynamic* karena atribut di dalamnya yang selalu berubah-ubah. Salah satu atribut yang selalu berubah adalah jumlah pengguna yang mengakses layanan *cloud*. Algoritma *load balancing* yang dikembangkan harus mampu mengakomodasi perubahan ini sehingga layanan tetap berjalan tanpa penurunan performa.

Teknologi *cloud computing* terdiri dari beberapa *server*, *data center*, *virtual machine*, perangkat penyimpanan dan lain-lain yang saling terhubung [11]. Saat ini sistem *cloud computing* menganut teknologi virtualisasi yang dianggap dapat meningkatkan efisiensi sumber daya dari *data center*. Hal ini dikarenakan keberadaan beberapa *Virtual Machine* (VM) dapat membantu untuk mengalokasikan tugas ke satu *physical server* [12]. Hal ini menyebabkan algoritma *load balancing* untuk mengatur pengalokasian tugas ke VM menjadi sangat krusial di dalam arsitektur *cloud computing*. Namun terkadang dalam mengalokasikan tugas dimungkinkan dihadapkan pada permasalahan keterbatasan sumber daya yang memadai. Seperti pada penelitian [13] yang mencoba mengembangkan algoritma *load balancing* yang menangani keterbatasan sumber daya pada perusahaan penyedia layanan kecil dan menengah.

Oleh karena itu, pada penelitian ini akan dicoba untuk meningkatkan performa dari algoritma *load balancing* pada VM. Penelitian ini akan berfokus untuk penanganan masalah keterbatasan sumber daya *physical server*. Metode yang diusulkan yaitu dengan menggabungkan metode yang sudah ada yaitu Equally Spread Current Execution (ESCE) dan Throttled. Hasil implementasi algoritma ini disimulasikan dengan bantuan CloudAnalyst-A CloudSim based Visual Modeler [14] yang merupakan pengembangan dari CloudSim [15] yang lebih dulu dikenal sebagai simulator di lingkungan *cloud computing*. Pada bagian ke 2 dan 3 paper ini akan membahas tentang beberapa penelitian terkait yang telah dikembangkan sebelumnya. Kemudian pada bagian 4 akan membahas tentang metode yang kita usulkan. Pada section 5 akan menyajikan hasil dari simulasi dan analisisnya. Pada bagian terakhir berisi tentang kesimpulan dari hasil yang didapatkan.

II. PENELITIAN TERDAHULU

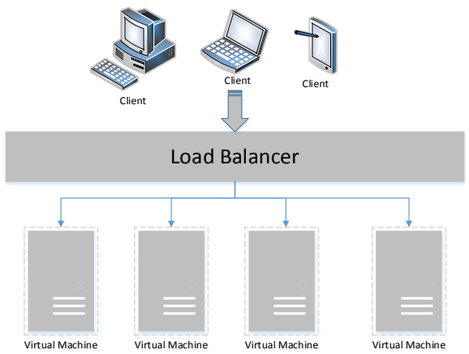
Pada bagian ini akan dijelaskan mengenai beberapa algoritma *load balancing* pada lingkungan *cloud computing* yang telah dikembangkan sebelumnya. Fokus utamanya yaitu dengan melakukan penjadwalan tugas ke VM yang ada dengan waktu respon seminimal mungkin. Di dalam lingkungan *cloud computing* terdapat beberapa atribut yang mempengaruhi proses *load balancing* diantaranya jumlah CPU, *bandwidth*, *memory*, serta perangkat penyimpanan. Jadi tujuan utama dari algoritma *load balancing* yaitu memilih untuk VM yang paling optimal berdasarkan nilai atribut-atribut di atas sehingga mendapatkan waktu respon seminimal mungkin. Terdapat beberapa algoritma yang telah dikembangkan sebelumnya untuk menangani masalah *load balancing* ini meskipun memang masih belum terlalu banyak penelitian di bidang ini.

Brototi et al. [16] mengembangkan algoritma *load balancing* dengan mengadopsi algoritma Local Search Sthocastic Hill Climbing. Algoritma ini berjalan secara sederhana dengan melakukan perulangan dan penambahan nilai suatu variabel. Perulangan ini akan berhenti pada kondisi yang disebut "peak" di mana sudah tidak ada tetangga yang memiliki nilai yang lebih optimal. Ternyata algoritma ini berpotensi terjadinya *local optimal*. Hasil dari algoritma ini juga tidak jauh berbeda jika dibandingkan dengan algoritma yang sudah ada seperti Round Robin dan FCFS. Kemudian selanjutnya [17] mengembangkan algoritma Genetika memperbaiki performa dari Sthocastic Hill Climbing. Dengan algoritma genetika dapat mencegah terjadinya local optimal dalam mencari VM yang paling optimum. Algoritma ini memodelkan atribut-atribut yang mempengaruhi proses *scheduling* ke bentuk kromosom yang merupakan representasi dari solusi. Selanjutnya dicari kromosom dengan nilai *fitness* tertinggi

untuk dipilih sebagai solusi yang paling optimal. Hasil performa dari algoritma ini lebih baik jika dibandingkan dari dengan algoritma Sthocastic Hill Climbing dan dua algoritma terdahulu lainnya yaitu Round Robin dan juga FCFS.

Selain memanfaatkan algoritma *soft computing*, juga telah dikembangkan metode *load balancing* pada lingkungan *cloud computing* dengan meningkatkan performa algoritma primitif yang telah ada. Seperti pada [11] yang memodifikasi algoritma Throttled dengan membagi tugas secara seragam. Tidak seperti pada Throttled yang selalu melakukan pengecekan dari *node* awal. Tetapi algoritma modifikasi ini membagi tugas secara merata dengan mengubah indeks nol dari VM secara *incremental*. Hasilnya, jika dibandingkan Throttled dan Round Robin lebih baik waktu responnya. Kemudian pada penelitian lain [18] dilakukan penggabungan antara algoritma Throttled dengan algoritma Equally Spread Current Execution. Penggabungan ini dimaksudkan untuk mendapatkan kelebihan dari kedua metode tersebut. Kelebihan Throttled yaitu tersedianya daftar ketersediaan VM, sedangkan ESCE memiliki daftar VM dan *queue* tugas yang ada. Sehingga dengan beberapa daftar tersebut proses *load balancing* diharapkan memberikan performa lebih bagus. Hasil dari penelitian ini menunjukan bahwa algoritma penggabungan yang dikembangkan memiliki waktu respon yang paling bagus dibanding dengan Round Robin, ESCE, dan juga Throttled. Namun, dari masing-masing penelitian di atas memiliki studi kasus yang berbeda-beda dalam melakukan simulasi. Hal ini menyulitkan ketika ingin melakukan perbandingan secara komprehensif karena dimungkinkan setiap algoritma memiliki spesialisasi khusus. Misalnya, suatu algoritma memnunjukan performa bagus ketika studi kasusnya memiliki *peak hour* yang panjang tetapi menurun ketika kasus memiliki *peak hour* yang pendek. Selain itu juga belum ada penelitian yang secara spesifik menangani *load balancing* dengan spesifikasi *physical hardware* yang rendah. Sehingga penelitian ini akan mencoba untuk menyajikan penjelasan yang mendalam baik metode maupun desain eksperimennya.

III. ALGORITMA LOAD BALANCING



Gambar 1. Konsep Load Balancing pada Komputasi Awan

Algoritma *load balancing* pada lingkungan *cloud computing* digunakan untuk menentukan VM mana yang akan digunakan untuk menjalankan *request* yang masuk. *Load balancing* membantu mengurangi penggunaan bandwidth yang berdampak pada berkurangnya biaya yang dikeluarkan dan memaksimalkan layanan yang ditawarkan oleh *service provider* [18]. Algoritma ini juga bertujuan untuk meminimalisasi waktu respon layanan yang diberikan kepada pengguna. Algoritma yang bisa diterapkan untuk melakukan *load balancing* dan sudah ada saat ini adalah algoritma FCFS (First Come First Serve), Round Robin, Equally Spread Current Execution (ESCE) dan Throttled. Tetapi pada penelitian ini hanya akan dibahas mengenai Round Robin, ESCE dan Throttled.

A. *First Come First Serve (FCFS)*

Algoritma ini adalah algoritma paling mudah yang digunakan untuk mengalokasikan *request* pada VM. FCFS mengalokasikan *request* sesuai dengan urutan masuknya request ke VM selanjutnya sesuai dengan urutan VM. Implementasi untuk algoritma ini dapat dengan mudah dilakukan dengan menggunakan FIFO *queue* [19].

B. *Round Robin (RR)*

Merupakan algoritma penjadwalan yang sudah lama dipakai untuk menjadwalkan tugas

di berbagai mesin pemroses data. Algoritma ini menggunakan konsep *quantum of time*. Setiap mesin diberikan sebuah interval waktu atau *quantum* untuk menjalankan operasinya. Waku *quantum* memegang peranan penting pada algoritma ini. Apabila waktu *quantum* terlalu besar, maka algoritma ini menjadi sama dengan algoritma FCFS. Sebaliknya apabila waktu *quantum* terlalu kecil, maka frekuensi pergantian prosesnya akan menjadi sangat tinggi [11]. Algoritma Round Robin termasuk dalam golongan *static load balancer*. Sehingga algoritma ini memiliki banyak kekurangan ketika diaplikasikan di lingkungan yang berubah-ubah seperti lingkungan *cloud computing*.

C. *Equaly Spread Current Excequtions (ESCE)*

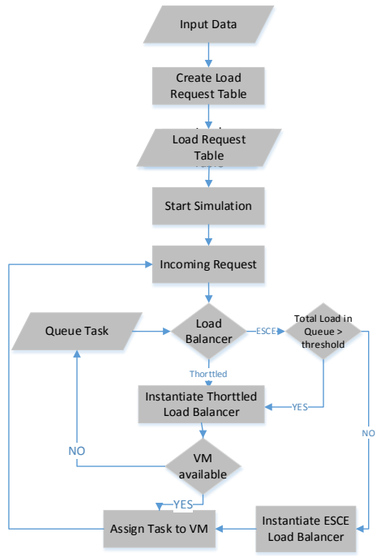
Load balancer ini menyimpan informasi dalam sebuah tabel indeks yang berisi Virtual Machine serta *request* yang sedang diproses oleh VM tersebut. Tugas dari *load balancer* adalah memastikan setiap VM menjalankan *request* dengan jumlah alokasi yang sama. Ketika terdapat *request* baru ke *data center*, *load balancer* akan membaca tabel indeks untuk mencari VM yang sedang menjalankan *request* dengan jumlah alokasi paling kecil. Setelah itu, *load balancer* mengirimkan identitas VM ke *data center* untuk kemudian mengalokasikan *request* tersebut ke VM yang telah dipilih dan melakukan *update* terhadap data jumlah alokasi VM tersebut [11]. Algoritma ini kurang cocok apabila diterapkan pada kasus dengan jumlah *request* tinggi dan *resource* yang terbatas, karena *load balancer* akan mengalokasikan *request* terhadap VM dengan alokasi terendah meskipun VM tersebut sedang menjalankan *request* yang lain. Akan tetapi, algoritma ini memiliki performa yang bagus ketika *load request* rendah.

D. *Throttled Load Balancing Algorithm*

Pada algoritma Throttled, data VM disimpan beserta status ketersediaanya (AVAILABLE/ BUSY) [11]. *Load balancer* bertugas untuk mencari VM mana yang tersedia ketika terdapat *request* yang masuk. Pencarian dilakukan dari VM pertama sampai ditemukan VM dengan status AVAILABLE. Apabila tidak ditemukan VM yang AVAILABLE, *load balancer* mengembalikan nilai -1 ke *data center*. Setelah itu, *data center* akan memasukkan *request* tersebut ke dalam *queue*. *Request* yang terdapat di dalam *queue* akan diproses ketika telah ada VM dengan status AVAILABLE. Algoritma ini kurang cocok apabila diterapkan pada kasus dengan jumlah *request* rendah dan *resource* yang terbatas, karena *load balancer* akan menunggu dan mencari VM yang tersedia untuk dialokasikan. Tetapi, algoritma ini cocok dengan *load request* yang sangat tinggi.

IV. METODE

Algoritma yang diajukan berusaha mengambil kelebihan dari algoritma load balancing Throttled dan ESCE sehingga algoritma ini dapat menangani kasus dengan jumlah *request* yang tinggi maupun rendah pada *resource* yang terbatas. Pada resource yang terbatas, VM memiliki kemampuan yang terbatas pula dalam menjalankan *request* yang masuk. Algoritma ESCE membagi jumlah *request* secara merata pada semua VM sehingga lebih efektif ketika jumlah *request* yang masuk rendah. Sedangkan pada algoritma Throttled, tabel indeks yang berisi status dari VM mampu membantu dalam mengalokasikan *request* pada VM yang sesuai. Sehingga waktu respon ketika jumlah *request* tinggi menjadi lebih baik.



Oleh karena itu, dibangun sebuah

algoritma yang dapat memfasilitasi pergantian algoritma *load balancer* sesuai jumlah *request* secara dinamis. Berbeda dengan penelitian sejenis yang juga memanfaatkan kelebihan dari kedua metode tersebut [18], metode yang kami usulkan berfokus pada lingkungan *cloud computing* dengan resource terbatas. Penelitian sebelumnya menggabungkan informasi status VM pada algoritma throttled dan informasi alokasi tiap VM pada algoritma ESCE menjadi satu *hashmap*. *Hashmap* yang berisi gabungan informasi ini digunakan untuk memilih VM yang akan dialokasikan pada tiap request yang masuk. Sedangkan, pada penelitian ini terlebih dahulu dibangun sebuah *request table* berisi informasi algoritma mana dari kedua algoritma tersebut yang akan digunakan pada waktu tertentu. Apabila *load balancer* yang digunakan adalah ESCE, dilakukan pengecekan menggunakan *Queue Thresholding*. Setelah itu, *load balancer* yang dipilih digunakan untuk mencari VM sesuai dengan algoritma masing-masing. *Request* yang masuk akan diproses oleh VM tersebut. *Hashmap* yang ada pada algoritma throttled digunakan untuk melakukan pengecekan secara dinamis pada alokasi dan antrian *request*. Apabila jumlahnya melebihi atau dibawah batas *threshold* yang telah ditetapkan, maka algoritma *load balancing* yang digunakan juga akan berubah secara dinamis. Proses dilakukan sampai semua *request* selesai diproses.

A. Load Request Table

Tabel ini dibangun untuk dapat mengetahui jumlah *request* setiap jam. Jumlah request (TR) tiap jam bisa diperoleh dari *Request per User per Hour* (RU), *Data Size per User* (DS), dan *Total User per Hour* (TU). *Total User per Hour* adalah total pengguna dari keseluruhan *userbase* pada jam tertentu dengan memperhitungkan *peak hour* dari setiap *userbase*. n merupakan jumlah *userbase* yang sedang *peak hour* dan i merupakan jumlah *userbase* yang tidak sedang pada jam *peak hour*. Jumlah dari n dan i sama dengan jumlah *userbase*. Persamaan untuk melakukan perhitungan jumlah *request* dan *Total User per Hour* dapat dilihat pada Persamaan (1) dan (2).

$$TR = RU * DS * TU \quad (1)$$

$$TU = \sum_n user_n + \sum_i user_i \quad (2)$$

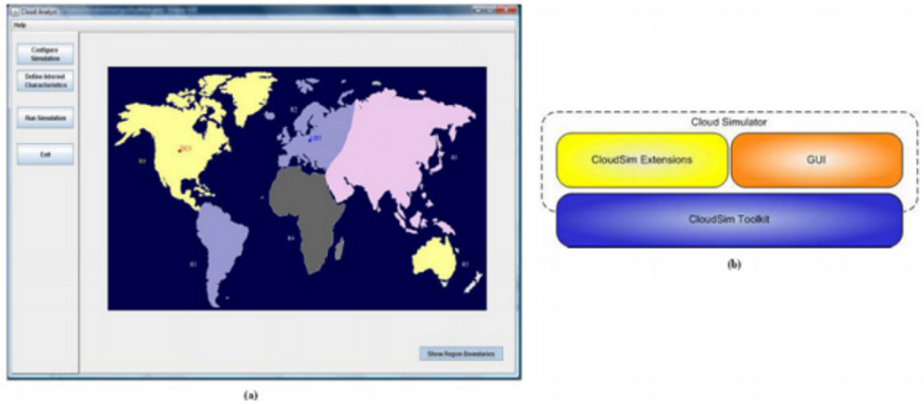
Dari data jumlah *request* untuk setiap jam selama 24 jam ditentukan *load balancer* apa yang akan digunakan pada jam tertentu sehingga membentuk sebuah jadwal. Untuk memilih *load balancer* digunakan sebuah nilai *threshold*. Nilai tersebut dibandingkan dengan jumlah *request* tiap jam. *Threshold* didapatkan dari 30% rata-rata jumlah *request*. Nilai tersebut dinilai paling efektif setelah dilakukan beberapa kali percobaan.

B. Queue Threshold

Pada saat proses transisi dari *load balancer* Throttled ke ESCE dimungkinkan terdapat sisa *request* di dalam *queue* yang cukup besar dan tidak dapat ditangani oleh algoritma ESCE. Hal ini akan mengakibatkan total waktu respon yang dihasilkan menjadi lebih lama. Rata-rata total request digunakan sebagai batas nilai yang dibandingkan dengan jumlah *load* dari *request* yang terdapat didalam *queue*. Sehingga sebelum dilakukan pergantian dari Throttled ke ESCE harus dipastikan dahulu bahwa sisa *request* di dalam *queue* berada di batas yang dapat diproses oleh algoritma ESCE.

V. DESAIN EKSPERIMEN

Eksperimen yang dilakukan bertujuan untuk menunjukkan performa algoritma yang diusulkan pada kasus request yang tinggi maupun rendah dengan jumlah resource yang terbatas. Eksperimen dilakukan dengan 2 skenario yang berbeda. Pertama yaitu dengan menggunakan physical hardware yang terbatas dengan membuat variasi di penggunaan VM yaitu 25, 50, dan 75. Kemudian untuk skenario yang kedua akan dicoba dengan meningkatkan physical hardware yang digunakan untuk mengetahui performa algoritma pada resource yang lebih memadahi. Pada skenario yang kedua ini juga dilakukan variasi penggunaan VM 25, 50, dan 75. Semua skenario menggunakan Service Broker Policy "optimize response time" meskipun tidak akan banyak berpengaruh karena hanya akan digunakan satu *data center*. Masing-masing skenario akan disimulasikan selama 24 jam.



Gambar 3. Cloud analyst

Konfigurasi yang digunakan dalam melakukan simulasi dapat dilihat pada tabel 1 dan tabel 2.

Tabel 1. Konfigurasi Userbases

Name	Region	Req per User per Hours	Data size per Request	Peak hour start	Peak hour end	Average peak users	Average off peak user
UB1	0	60	100	13.00	15.00	400000	400
UB2	1	60	100	15.00	17.00	100000	100
UB3	2	60	100	20.00	22.00	300000	300
UB4	3	60	100	06.00	08.00	150000	150
UB5	4	60	100	21.00	23.00	50000	50
UB6	5	60	100	09.00	15.00	80000	80

Tabel 2. Konfigurasi Data Center

Parameter	Value Used
VM Image Size	10000
VM Memory	1024 Mb
VM Bandwidth	1000
Data Center Architecture	X86
Data Center OS	Linux
Data Center VMM	Xen
Data Center Number of Machine	20
Data Center Memory per Machine	2048 Mb
Data Center Storage per Machine	100000
Data Center Bandwidth per Machine	10000
Data Center Number of Processor per Machine	4
Data Center Processor Speed	100 MIPS
Data Center VM Policy	Time Shared
User Grouping Factor	1000
Request Grouping Factor	100
Executable Instructions Length	250

Pada penelitian yang berkaitan dengan *cloud computing,* dibutuhkan *resource* yang sangat mahal untuk implementasi serta uji coba. Sehingga para peneliti akan sangat terbantu dengan adanya suatu *tools* yang dapat mensimulasikan lingkungan *cloud computing*. *Tools* tersebut juga harus dapat merepresentasikan kondisi sesungguhnya pada kenyataan. Pada penelitian ini sendiri akan digunakan *tools* yang bernama Cloud Analyst dalam menguji *load balancer* yang diusulkan. Cloud Analyst merupakan *tools* simulasi untuk lingkungan *cloud computing* dalam skala besar [13]. Cloud Analyst dikembangkan berdasarkan CloudSim [14] dengan menambahkan fitur-fitur konfigurasi yang membuat simulasi lebih realistis. Selain itu *tools* simulasi ini juga dilengkapi dengan GUI yang memudahkan pengguna untuk menggunakannya.

Cloud Analyst memberi fasilitas kepada pengguna untuk dapat mengembangkan metodenya secara programmatically. Salah satunya yaitu dengan mengembangkan algoritma load balancing. Namun, juga dimungkinkan untuk mengembangkan Service Broker Policy. Pada Cloud Analyst terdapat beberapa kondisi buatan yang disesuaikan dengan keadaan sebenarnya. Misalnya seperti pembagian seluruh belahan dunia menjadi 6 wilayah benua. Setiap benua menggambarkan satu basis pengguna. Cloud Analyst menggunakan satu *constraint* waktu yaitu GMT. Sehingga masing-masing waktu pada *region* yang berbeda akan dikonversi ke GMT. Selain itu terdapat *constraint* mengenai *peak hour* dan *off peak hours* di mana yang membedakan adalah jumlah pengguna yang mengakses. Di samping itu masih banyak lagi nilai-nilai lain yang dapat ditentukan sesuai

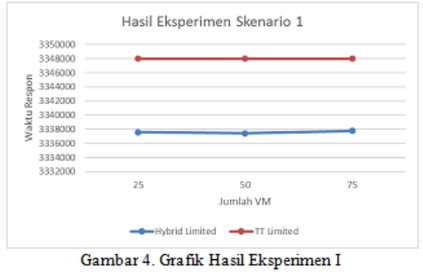
dengan model simulasi yang diinginkan.

VI. HASIL EKSPERIMEN

Pada skenario yang pertama, ketiga percobaan menunjukkan hasil yang sama, yaitu metode yang diajukan dapat menghasilkan waktu respon yang lebih cepat dari pada menggunakan algoritma Throttled. Rata-rata perbedaan waktu responnya sebesar 10381.76 ms. Hal ini terjadi karena ketika algoritma throttled digunakan pada jumlah request yang rendah, load balancer menunggu sampai ada VM yang AVAILABLE. Sedangkan dengan menggunakan algoritma hybrid, request tersebut dapat langsung dialokasikan menggunakan algoritma ESCE. Waktu respon yang paling baik dihasilkan dengan menggunakan metode hybrid dengan jumlah virtual machine sebanyak 50 yaitu sebesar 3337424.26 ms. Tabel 3 dan Gambar 4 menunjukkan perbandingan nilai waktu respon antar kedua metode pada tiga kondisi. Jumlah VM dibedakan menjadi 25, 50, dan 75 VM untuk masing-masing percobaan dengan konfigurasi *userbase* dan *data center* yang sama. Hasil eksperimen menunjukkan bahwa jumlah VM yang banyak tidak mengakibatkan waktu respon menjadi lebih cepat dan sebaliknya. Hal ini dikarenakan jumlah dan spesifikasi *physical hardware* yang terbatas. Hasil ini menunjukan bahwa algoritma yang dikembangkan pada penelitian ini mampu unggul dibanding algoritma Throttled sendiri di spesifikasi *hardware* yang terbatas. Skenario 1 ini juga telah dicoba menggunakan algoritma ESCE dan Round Robin. Akan tetapi, waktu simulasinya sangat lama sehingga tidak dapat diperoleh hasil simulasinya.

Tabel 3. Hasil Eksperimen I

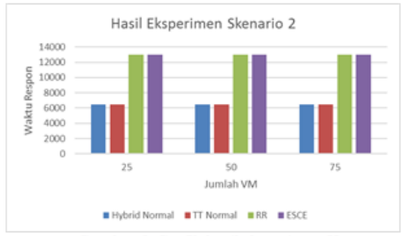
Jumlah VM	RT Menggunakan Hybrid (ms)	RT Menggunakan Throttled (ms)
25	3337576.83	3347978.34
50	3337424.26	3347978.21
75	3337788.14	3347978.01



Gambar 4. Grafik Hasil Eksperimen I

Tabel 4. Hasil eksperimen II

Jumlah VM	Jumlah Physical Hardware	RT Menggunakan Hybrid (ms)	RT Menggunakan Throttled (ms)	RT Menggunakan RR (ms)	RT Menggunakan ESCE (ms)
25	**20**	6482.29	6482.25	12977.01	12977.37
50	**20**	6482.29	6482.26	12977.0	12977.38
75	**20**	6482.30	6482.32	12976.94	12977.27



Gambar 5. Grafik hasil eksperimen II

Eksperimen yang kedua dilakukan dengan menambahkan jumlah *physical hardware* menjadi 20. Hal ini dilakukan untuk mengetahui kemampuan algoritma yang dibangun pada kondisi jumlah *resource* yang normal. Hasil eksperimen dapat dilihat pada Tabel 4 dan Gambar 5. Skenario 2 ini juga dijalankan pada 25, 50, dan 75 VM. Hasilnya menunjukkan bahwa pada kondisi dengan *physical hardware* yang mencukupi, algoritma yang dibangun mampu memberikan waktu respon yang tidak jauh berbeda dan cenderung lebih stabil dari algoritma Throttled. Algoritma RR dan ESCE menghasilkan waktu respon kurang lebih dua kali lipat dibandingkan dengan algoritma hybrid dan Throttled.

VII. KESIMPULAN

Teknologi *cloud computing* saat ini semakin banyak digunakan dalam menyediakan layanan yang berkaitan dengan perangkat lunak. Beberapa layanan-layanan seperti Facebook, Twitter, Path, dan masih banyak lagi merupakan contoh aplikasi yang memanfaatkan layanan *cloud computing*. Layanan *cloud computing* sangat erat kaitannya dengan kebutuhan akan *reliability* yang tinggi. Karena layanan ini harus tetap tersedia disaat memang dibutuhkan oleh pengguna. Kebutuhan *reliability* yang tinggi ini berimbas pada kebutuhan *hardware* yang memadai. Hal ini menjadi masalah pada perusahaan yang masih berkembang. Karena kebutuhan spesifikasi yang memadai akan membutuhkan biaya yang tinggi.

Kemudian, *realibility* pada *cloud computing* juga erat kaitannya dengan proses *load balancing*. *Load balancing* pada *cloud computing* dilakukan dengan tujuan meminimalkan waktu respon. Berangkat dari kedua kebutuhan itu, pada penelitian ini dicoba untuk mengembangkan algoritma *load balancing* baru untuk lingkungan *cloud computing* yang cocok untuk spesifikasi *hardware* yang terbatas. Algoritma ini dikembangkan dengan menggabungkan algoritma yang sudah ada yaitu ESCE dan Throttled.

Dari hasil pengujian menunjukan bahwa algoritma yang dikembangkan dalam metode ini menghasilkan waktu respon yang lebih baik dibanding algoritma Throttled pada hardware yang terbatas. Selisih waktu respon yang dihasilkan juga cukup signifikan. Kemudian algoritma ini juga diuji pada spesifikasi *hardware* yang lebih memadai. Hasilnya menunjukan bahwa waktu respon yang dihasilkan tetap tidak jauh berbeda dibandingkan dengan algoritma Throttled dan lebih baik dibandingkan dengan algoritma RR dan ESCE. Sehingga dapat diambil kesimpulan bahwa algoritma yang dihasilkan pada metode ini memiliki performa yang konsisten. Akan tetapi, pada penelitian selanjutnya perlu dilakukan peningkatan metode lagi sehingga algoritma ini dapat memberikan hasil yang signifikan baik di spesifikasi *hardware* yang terbatas maupun yang sudah memadai. Perbaikan sendiri dapat dilakukan dari sisi manajemen transisi yang lebih baik. Karena dari penelitian ini sendiri masih ada kekurangan pada saat transisi perubahan dari algoritma Throttled ke algoritma ESCE. Selain itu, perbaikan juga dapat dilakukan dengan metode penentuan nilai *threshold* yang lebih baik. Sehingga dengan *threshold* yang tepat diharapkan memberikan hasil yang lebih efisien.

DAFTAR PUSTAKA

[1] Arbrust and Michael , "A view of cloud computing," in *Communication of te ACM 53.4*, 2010.

[2] C. o. Weinhardt, B. Blau and J. Stober, "Cloud Computing- A Classification, Business Models, And Research Directions," *Bussines and Information System Engineering,* vol. 5, pp. 391-399, 2009.

[3] Buya and Rajkumar , "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer System,* pp. 599-616, 2009.

[4] Dikaiakos and D. Marios, "Cloud computing: distributed internet computing for IT and scientific research," *IEEE Internet Computing*, pp. 10-13, 2009.

[5] C. Yahsueh and W. Mingchang, "Understanding the determinants of cloud computing adoption," in *Journal of Industrial Management and Data System*, 2011.

[6] G. Garrison, K. Sanghyun and R. L. Wakefield, "Success factor for deploying cloud computing," *Communication of the ACM,* vol. 55, no. 9, pp. 62-68, 2011.

[7] Z. Caczko and V. Mahadevan, "Availability and Load Balancing in Cloud Computing," *IPCSIT,* vol. 14, 2011.

[8] A. Vouk, "Cloud computing- issues, research and implementations," *In Proc. of Information Technology Interface,* pp. 31-40, 2008.

[9] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., 1979.

[10] Rimal and B. Prasad, "A taxonomy and survey of cloud computing systems.," in *5th International Joint Conference on INC, IMS, and IDC*, 2009.

[11] Domanal and Shridal , "Load Balancing in Cloud Computing using Modified Throttled Algorithm," in *Cloud COmputing in Emerging Market*, 2013.

[12] G. Hadi and M. Gashemazar , "SLAbased Optimization of Power and Migration Cost in Cluser, Cloud, and Grid Computing," in *12th International Simposium on Cloud*, 2012.

[13] L. Guo and Y. Guo, "Real time elastic cloud management for limited resources," in *IEEE International Conference on Cloud Computing*, 2011.

[14] Wickremashinghe, Bathiya and Rodrigo. N, "Cloud analyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications," *24th AINA,* 2010.

[15] Calherios and Rodrigo N., "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience,* vol. 41, no. 1, pp. 23-50, 2011.

[16] Moundal, Brototi and K. Dasgupta, "Load balancing in cloud computing using stochastic hill climbing-a soft computing approach," in *Procedia Technology*, 2012.

[17] K. Dasgupta, "A Genetic Algorithm (GA) based Load Balancing Strategy for Cloud Computing," *Procedia Technology,* vol. 10, pp. 340-347.

[18] V. Bhagwaiya, "Hybrid Approach using Thorttled and ESCE Load Balancing Algorithm in Cloud Computing," in *ICGCCEE*, 2014.

[19] S. Mohapatra and K. S. Rekha, "A Comparison for Four Popular Heuristic for Load Balancing of Virtual Machine in CLoud Computing," *International Journal of COmputer Application,* vol. 68, no. 6, 2013.