Sistem *Load Balancing* Menggunakan *Least Time First Byte* dan *Multi Agent System*

Muhammad Faizal Afriansyah[1], Maman Somantri[2], Munawar Agus Riyadi[3]

***Abstract*—Network activity has increased in every year due to rapid growth of internet users. This phenomenon eventually increases the load server. The high load on server makes server down. The proposed system to handle the issue is using Least Time First-Byte algorithm and multi-agent system in distributed load balancing. The agent collects information resources on the backend servers and communicates with the agents. The Least Time First-Byte algorithm is then combined with the information resources from the agent, called as Least Time First Byte with Multi Agent System (LFB-MAS). The simulation results show that LFB-MAS performs load balance efficiently to all server and provides better performance. The LFB-MAS can process 100% from the 1,800 requests, whereas WLC algorithm is only capable of processing 74.50% from 1,800 requests and LFB without agent is only capable of processing 75.61% from 1,800 requests. The results prove that LFB-MAS can handle high tasks or requests and is reliable.**

***Intisari*—Lalu lintas jaringan meningkat setiap tahunnya dikarenakan cepatnya peningkatan pengguna internet. Hal ini dapat meningkatkan beban kerja *server*. Beban yang tinggi pada *server* akan menyebabkan *server* menjadi *down*. Metode yang diajukan untuk menangani masalah tersebut adalah dengan *load balancing* terdistribusi menggunakan algoritme *least time first-byte* dan *multi agent system*. *Agent* akan mengumpulkan informasi dan dapat berkomunikasi antar *agent* sesuai tugas yang diberikan. Metode *load balancing* yang diajukan dapat disebut dengan *Least Time First Byte with Multi Agent System* (LFB-MAS). Berdasarkan data yang diperoleh pada hasil pengujian, metode WLC hanya mampu memproses sekitar 74,50% dari 1.800 *request* dan LFB tanpa menggunakan *agent* hanya mampu memproses 75,61% dari 1.800 *request*. Metode LFB-MAS dapat memproses 100% dari semua *request* atau 1.800 *request* yang diberikan. Hal ini menunjukkan bahwa LFB-MAS dapat melakukan sistem *load balancing* dengan efisien ke seluruh *server backend* yang tersedia dengan hasil yang lebih baik, tidak ada *bottleneck, low risk server overload*, dan andal.**

***Kata Kunci*-- Andal, *server*, JADE, *multi agent system*, *load balancing*.**

I. PENDAHULUAN

Server menjadi hal yang sangat penting dan diperlukan pada era teknologi saat ini. *Server* menjadi pusat pelayanan dari seluruh pengguna. *Server* itu sendiri merupakan sebuah sistem komputer dengan spesifikasi sumber daya yang sangat tinggi daripada komputer umumnya dan bertugas menyediakan jenis layanan tertentu serta pengolah data yang sangat kompleks dan besar. Semakin banyak pengguna, maka semakin kompleks juga infrastruktur *server* yang dibangun. Infrastruktur *server* yang dibangun harus dapat menampung sejumlah *request* yang datang dari para penggunanya. *Load server* akan tinggi seiring dengan banyaknya penggunaan layanan yang disediakan oleh *server*. Beban *server* yang tinggi mengakibatkan kecepatan proses menurun dan bisa membuat *server* berhenti bekerja. Berdasarkan masalah tersebut, diperlukan teknologi yang sesuai untuk dapat mengatasi permasalahan ini. Sistem terdistribusi digunakan dalam infrastruktur untuk mendistribusikan sejumlah *request* yang masuk dari pengguna kepada *server-server* untuk diproses. Selain sistem terdistribusi, untuk dapat menangani masalah tersebut secara andal dapat ditambahkan sebuah mekanisme *load balancing*. *Load balancing* secara terdistribusi banyak dipraktekkan oleh sejumlah ilmuwan [1]--[4]. *Load balancing* mutlak diperlukan untuk meningkatkan reliabilitas sebuah layanan dengan banyaknya pengguna dengan skala besar. Pada sistem terdistribusi, *load balancing* merupakan sebuah metode untuk membagi beban secara merata di semua unit komponen (*server*) [5]. *Load balancing* sangat dibutuhkan pada proses skala besar. Mekanisme pada *load balancing* adalah mendistribusikan sumber daya (*resources*) untuk setiap permintaan pelanggan (pengguna) pada *server*, sehingga tidak ada *node server* yang *overload* atau tetap *idle*. Mekanisme *load balancing* dapat dikombinasikan dengan *multi agent system*. *Load balancing* yang berfokus pada lingkungan *cloud computing* dengan menggunakan *agent* telah dianalisis pada sebuah penelitian [6]. Metode *load balancing* menggunakan *multi agent system* dapat menciptakan sistem yang andal dan efisien.

Multi agent system merupakan sebuah perangkat lunak berbasis sistem terdistribusi menggunakan *agent* dengan kemampuan khusus dan *autonomous* (mandiri) pada sebuah jaringan [3]. *Software agent* akan terus-menerus melakukan tugas yang telah diberikan oleh pengguna dalam lingkungan khusus [7]. *Agent* memiliki kemampuan atau karateristik seperti kooperatif, mandiri, dapat berkomunikasi antar *agent*, dan bergerak *(mobile)* [8]. *Agent* dapat berpindah dari satu *node* ke *node* yang lain dengan bebas sesuai dengan tugas yang diberikan.

Desain mekanisme *load balancing* menggunakan algoritme *Least Time First Byte* dan *Multi Agent System (LFB-MAS)* secara terdistribusi dibahas pada makalah ini. *Agent* bergerak ini memonitor keadaan *resources* dari *server backend*. Para *agent* berkomunikasi melalui *node server* dalam kelompok

[1]*Mahasiswa, Departemen Teknik Elektro Fakultas Teknik Universitas Diponegoro, Jl.Prof.H.Soedarto, S.H., Tembalang, Kota Semarang, Jawa Tengah 50275 INDONESIA (tlp: (024)-7460012; fax: (024)-7460012; e-mail: afriansyahfaizal@gmail.com)*

[2, 3] *Dosen, Departemen Teknik Elektro Fakultas Teknik Universitas Diponegoro, Jl.Prof.H.Soedarto, S.H., Tembalang, Kota Semarang, Jawa Tengah 50275 INDONESIA (tlp: (024)-7460012; fax: (024)-7460012; e-mail: mmsomantri@live.undip.ac.id, munawar@elektro.undip.ac.id)*

jaringan dalam melakukan tugasnya untuk mendapatkan informasi *resources*. Dengan integrasi metode ini, permintaan untuk dapat mencapai *resources* yang diinginkan lebih cepat dan *node* padat dapat diidentifikasi lebih efisien [9]. Sistem *load balancing* pada makalah ini menggunakan algoritme *load balancing Least Time First Byte* (LFB) dan teknologi JAVA yang dinamakan JADE [10]--[13]. Pemrograman JADE sama seperti pemrograman *aglets*, yaitu digunakan untuk mencari informasi kondisi dari *server*, terdapat *load* yang besar atau masih *idle*. JADE merupakan perkembangan dari pemrograman *agent* bergerak *aglets*. Informasi tersebut dikirim kembali ke *server* dan diolah. *Multi agent system* ini bersifat *cross platform* sehingga dapat digunakan di manapun (OS apapun) karena berbasis JAVA. Terdapat juga algoritme *load balancing* yang membagi beban yang diterima dengan melihat informasi kondisi dari *server* tersebut. Dengan demikian, terdapat satu *server* pusat yang bertugas untuk mengatur beban yang masuk ke dalam *server-server* agar tidak *overload* atau *idle* dan memantau kinerja *agent*. *Server load balancing* dapat melakukan sistem *load balancing* dengan efisien ke seluruh *server backend* yang tersedia, tidak ada *bottleneck*, *low risk server overload*, dan mempercepat waktu respons.

Secara keseluruhan, penulisan makalah ini terbagi menjadi beberapa bagian. Bagian II mendiskusikan penelitian sebelumnya terhadap penelitian yang dilakukan. Bagian III membahas tentang metodologi pada penelitian yang dilakukan. Bagian IV merupakan hasil dari uji penelitian yang berupa data tabel dan grafik dan analisis. Bagian V berisi kesimpulan dari penelitian.

II. ALGORITME *LOAD BALANCING* TERDISTRIBUSI MENGGUNAKAN *AGENT*

Bagian ini memaparkan gambaran singkat penelitian terdahulu yang telah dilakukan pada algoritme *load balancing* terdistribusi menggunakan *agent*. Bagian ini juga membahas perbedaan dari penelitian sebelumnya dan permasalahan yang terjadi. Penjelasan singkat metode pendekatan yang diajukan dipaparkan pada bagian ini.

Sebelumnya, banyak metode *load balancing* dengan menggunakan *agent* yang dikembangkan. Metode *agent* ini merupakan sebuah implementasi *load balancing server* secara dinamis menggunakan *agent. Agent* tersebut mengumpulkan informasi dengan tujuan tertentu sesuai dengan program yang ditulis di dalamnya. *Load balancing* dinamis secara terdistribusi sangat dibutuhkan untuk dapat meningkatkan kinerja dan keandalan sebuah sistem dengan skala besar [14]. Sebuah sistem terdistribusi biasanya mendistribusikan unit komputasi seperti *resources* yang terkoneksi dengan jaringan untuk dapat memenuhi kebutuhan skala besar dan kinerja tinggi [15]. Sistem terdistribusi banyak digunakan pada *multi agent system*. Sistem terdistribusi juga dapat mengefisienkan kinerja *multi agent system* pada aplikasi skala besar. Secara garis besar, sistem terdistribusi akan mendistribusikan *resources* yang ada pada unit komputasi (*server*) dan *load balance* akan menyeimbangkan *resources* yang telah digunakan agar tetap andal. *Agent* ini berkomunikasi untuk memonitor dan menjaga sistem tetap berjalan dengan baik. Setiap *agent* saling berkomunikasi dan bekerja sama dalam menjalankan tugas yang diberikan. *Agent* secara otomatis mengerjakan segala tugas yang diberikan tanpa ada campur tangan dari sistem lainnya. Terdapat beberapa rangkuman yang menjelaskan *load balancing* secara terdistribusi yang telah dilakukan oleh penelitian sebelumnya.

Sistem *cluster* merupakan sistem yang memberikan skalabilitas kepada setiap *server* baru yang akan ditambahkan [16]. Sistem *cluster* dibagi menjadi beberapa bagian, yaitu komputasi *cluster* dengan kinerja komputasi yang tinggi, komputasi *cluster* dengan ketersediaan yang tinggi, dan komputasi *cluster* dengan distribusi beban sesuai dengan tujuan dari sistem komputasi. Sistem cluster *web* merupakan sistem komputasi *cluster* dengan pembagian beban dapat mendistribusikan sejumlah *request web* ke sejumlah *server*. Konsep pembagian beban yang diperkenalkan adalah peningkatan penjadwalan dari algoritme *weight least connection* (WLC). Algoritme ini merupakan campuran dari algoritme *load balancing least connection* dan *weight*. Peningkatan algoritme penjadwalan WLC diajukan dengan menambahkan penjadwalan penonaktifan *server* dan pengaktifan *server* baru yang ditambahkan dalam daftar untuk menjaga keseimbangan beban antar *server*.

Load balancing adalah teknik untuk meningkatkan sumber daya, memanfaatkan penggunaan paralelisme, peningkatan *throughput*, dan untuk memperpendek waktu respons melalui distribusi yang tepat dari aplikasi [17]. *Load balancing* adalah teknologi aktif yang menyediakan seni membentuk, mengubah, dan menyaring lalu lintas jaringan, kemudian mengalihkan dan membagi beban ke *node* yang optimal. Konsep *load balancing* yang diajukan menggunakan *agent* untuk menyeimbangkan beban sistem dengan berdasarkan *threshold*. *Threshold* yang diajukan dengan mengukur nilai setiap *node server* untuk menentukan *server* tersebut termasuk kelebihan beban (*overloaded*) atau sedikit beban (*under loaded*). Sebuah *node server* dikatakan *node* yang berat ketika *node* tersebut melebihi *threshold* yang telah didefinisikan. Peningkatan jumlah *node* yang kelebihan beban pada sistem akan memengaruhi kinerja sistem. *Agent* bertugas untuk mengumpulkan informasi beban yang terdapat di semua *node server* dan memonitor kondisi beban kerja semua *node server*.

Dalam memecah atau membagi sebuah beban pada *web server*, umumnya *server cluster* dikonfigurasi untuk mendistribusikan *request* akses atau *server mirror* didistribusikan secara tata letak yang sesuai pada jaringan yang berbeda [18]. Terdapat bermacam-macam jenis kebijakan *load balancing* dan salah satunya adalah *Message Passing Interface* (MPI). Pendekatan *agent* memiliki ciri fleksibilitas tinggi, efisien, trafik jaringan yang rendah, *latency* komunikasi yang rendah, dan juga tingkat asinkron yang tinggi. Konsep *load balancing* yang digunakan adalah dengan menggunakan teknologi *agent* untuk melakukan *load balancing* dinamis, yaitu ketika *node server overload*, *agent* akan berpindah ke *node server* yang memiliki *resources* atau utilitas rendah untuk berbagi beban kerja dengan *server overload*. Kerangka kerja *load balancing* yang diajukan disebut *Platform Load Balancing* (PLB). *Agent* PLB akan

berkomunikasi antar *agent node server* untuk memperoleh informasi. *Agent* bisa menjadi hak milik *server* yang membuatnya dan melakukan perintah secara langsung untuk pemiliknya. *Agent* juga bisa berbagi antar kelompok *server*. *Agent* dapat berinteraksi menggunakan teknik *stigmergy*, yaitu *agent* dapat mengumpulkan informasi dari jejak tertinggal di lingkungan satu sama lain. *Agent* dapat mengumpulkan informasi yang ditempatkan pada *server* dengan *agent* lain yang sebelumnya telah berkunjung ke sana. *Stigmergy* adalah metode yang secara tidak langsung digunakan untuk interaksi antar *agent* sehingga dapat mengurangi lalu lintas jaringan dan mengambil keputusan yang cepat.

III. METODOLOGI

A. Gambaran Umum Sistem

Sistem *load balancing* pada lingkungan terdistribusi yang diujikan menggunakan *multi agent system* dengan algoritme *load balance* yang diajukan. Semua *multi agent system* dijalankan menggunakan JADE dengan JDK 1.8 dan aplikasi *load balancing* dijalankan menggunakan Nginx. Secara keseluruhan, sistem terdiri atas satu mesin *server* fisik (*server host*) dengan spesifikasi prosessor enam *core* AMD Phenom II X6 1055T, 16 GB RAM, dan keseluruhan tiga *T harddisk*. Server *host* terbagi menjadi lima *server* virtual. Empat *server* virtual berjalan pada sistem operasi Linux Debian 3.16 dan satu *server* virtual berjalan pada sistem operasi Windows 2012 Standard. Sistem virtual ini menggunakan perangkat lunak PROXMOX berbasis Linux Debian 3.16.

Server *host* berjalan pada sistem operasi debian Jessie (Kernel 3.16) dengan konfigurasi dan spesifikasi mesin virtual seperti pada Tabel I.

TABEL I
KONFIGURASI DAN SPESIFIKASI MESIN VIRTUAL

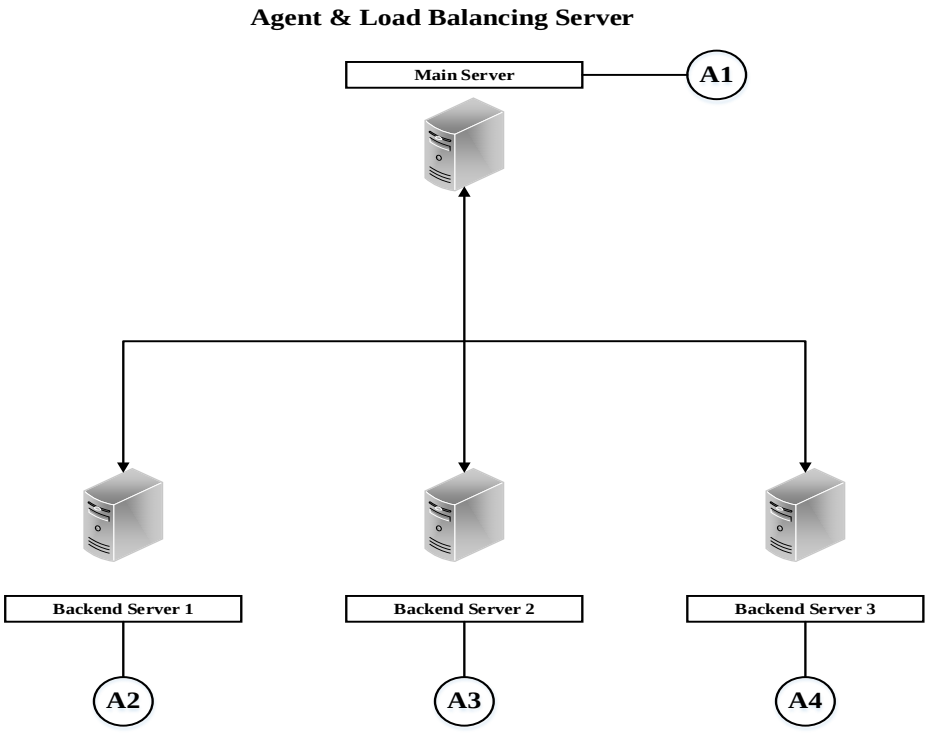
No	CPU	RAM	*Storage*	Keterangan
1	1 *core*	1 GB	10 GB	*Load balancing and agent server*
2	1 *core*	1 GB	10 GB	*Backend server 1*
3	1 *core*	1 GB	10 GB	*Backend server 2*
4	1 *core*	1 GB	10 GB	*Backend server 3*
5	2 *core*	2 GB	50 GB	*Client server*

Seluruh mesin *virtual* tersebut terkoneksi dengan LAN virtual pada PROXMOX dengan konfigurasi *bridge* dari setiap NIC mesin virtual. Pada mesin *virtual load balancer* terdapat dua NIC yaitu NIC *public* dan NIC *private*. NIC *public* ini mengarah langsung dengan pengguna/pengguna, sedangkan NIC *private* digunakan untuk interkoneksi dengan *server–server backend*. *Server-server backend* berisi konten aplikasi *web* dengan *standard CMS Wordpress*. Gbr. 1 merupakan gambaran sederhana topologi sistem yang diimplementasikan.

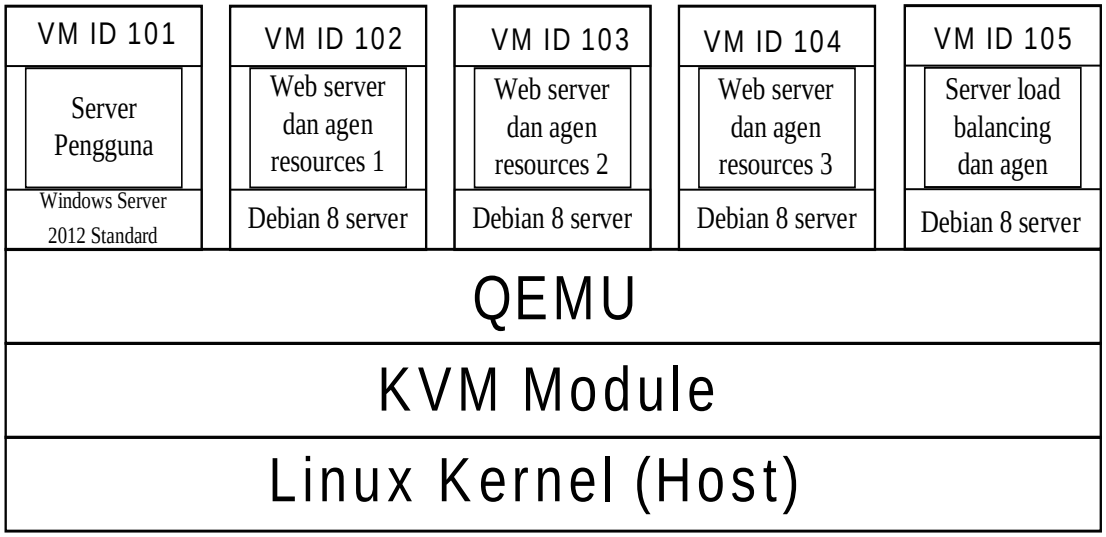
Gbr. 1 menjelaskan bahwa sistem *load balancing* penelitian ini terbagi menjadi dua bagian, yaitu *main server* dan *backend server*. *Main container* bertugas sebagai pusat informasi dan data dari para *agent* serta sebagai *load balancing server*. *Backend server* sebagai *server backend* dari layanan yang diberikan dan memproses *request* yang diberikan. *Main server* menerima *request* layanan dari pengguna yang kemudian diteruskan kepada *backend server* untuk diolah. Pada penelitian ini, algoritme yang digunakan untuk *load balancing* adalah *least time first byte* (LFB). Cara kerja dari algoritme *least time first byte* adalah membagi beban *request* layanan kepada *backend server* berdasarkan koneksi yang paling sedikit dan melakukan pengecekan *byte* pertama pada *server backend*. *Server* dengan nilai terkecil menjadi prioritas dalam melakukan *load balancing*. *Main server* juga mengirimkan *agent* kepada *backend server* untuk mengetahui informasi *resources* dari *server backend*. Agent mencatat dan memberikan kembali ke *main server*. *Main server* mengolah data informasi tersebut dan menjadikan dasar untuk melakukan *load balancing*. Beban yang ada dapat dibagi secara merata pada semua *backend server* sehingga menjadikan layanan yang efisien, mempertahankan *request* layanan dan memiliki waktu respon yang cepat dengan metode ini.

B. Cara Kerja Sistem

Cara kerja sistem dibutuhkan dalam menggambarkan proses–proses yang terjadi pada sistem. Cara kerja sistem menunjukkan alur proses sistem *load balancing* pada saat pertama kali *request* pengguna/pengguna masuk hingga diproses oleh sistem. Terdapat dua jenis *server*, yaitu *server* utama dan *server backend*. *Server* utama merupakan *server load balancing* dan *agent*. *Server backend* merupakan *server* aplikasi yang memproses *request* pengguna dan disajikan kepada pengguna. Proses yang pertama dilakukan adalah meneruskan *request* dari pengguna sejumlah n *request* ke *server backend* sesuai algoritme LFB. Kemudian, *server* utama mengirimkan *agent* ke *server backend*. *Agent server* utama dikonfigurasi untuk meminta informasi *resources* dari setiap *server backend*. Setelah *agent* sampai pada *server backend*, *server backend* memproses permintaan dari *agent server* utama. Informasi *resources* tersebut dikirimkan ke *server* utama melalui *agent server backend*. *Agent server backend* berisi informasi *resources server backend* dan dikirimkan ke *server* utama. *Server* utama memutuskan kondisi *load server backend* layak untuk melakukan layanan atau tidak berdasarkan informasi yang diberikan oleh *agent server backend* sesuai dengan parameter yang disetujui. *Server* utama memasukkan hasilnya ke dalam daftar *load balancing* dengan kondisi "normal" atau "*overload*". Kondisi "normal" merupakan kondisi ketika *server backend* layak untuk melakukan layanan. Kondisi dikatakan normal jika *load server* tidak melebihi *threshold*. Kondisi "*overload*" merupakan kondisi ketika *server backend* tidak dapat melakukan layanan karena *resources load server backend* tinggi atau melebihi *threshold*. *Threshold* yang diajukan pada penelitian ini adalah dengan mengukur *load CPU* dan *memory*. *Threshold* untuk *load CPU* sebesar 90% dan *load memory* sebesar 90%. *Server* utama mengirimkan *agent* kembali ke *server* secara berulang untuk memonitor kondisi sumber daya *server*. *Server backend* ditunda sementara sebanyak 1 s dari daftar *load balancing* jika kondisi *server backend overload*, sampai kondisi normal kembali. Selanjutnya, *server* utama menghitung *request* dengan algoritme LFB yang diajukan. *Server backend* memproses *request* dari pengguna tersebut dan memberikan hasilnya kepada pengguna melalui *server* utama. Pengujian dilakukan dengan perbandingan dengan algoritme dari penelitian sebelumnya dengan metode LFB-MAS yang diajukan.



Gbr. 1 Topologi sistem *load balancing.*



Gbr. 2 Arsitektur sistem *load balancing.*

C. Arsitektur Server

Arsitektur sistem diperlukan agar informasi pada sistem yang dibuat dapat dengan mudah dipahami. Arsitektur sistem yang digunakan dalam *load balancing* menggunakan metode LFB-MAS adalah arsitektur mesin virtual. Arsitektur ini terdiri atas perangkat keras, implementasi mesin virtual, nama mesin virtual, dan proses.

Gbr. 2 menunjukkan arsitektur sistem *load balancing* yang dibangun secara *virtual* pada komputer. Arsitektur ini terdiri atas komputer sebagai perangkat keras, aplikasi PROXMOX sebagai implementasi mesin virtual, lima mesin virtual, dua sistem operasi, dan proses pada masing masing mesin virtual. Mesin *virtual* yang digunakan pada arsitektur ini adalah *server load balancing* yang bertugas sebagai *server* dalam membagi beban dan sebagai *server agent* untuk mendapatkan informasi *resources* dari setiap *server backend*. *Server backend* 1 bertugas sebagai *web server* 1 untuk menyajikan data atau informasi kepada pengguna dan sebagai *agent* mengambil informasi resources dari *server backend* 1. *Server backend* 2 bertugas sebagai *web server* 2 untuk menyajikan data atau informasi kepada pengguna dan sebagai *agent* mengambil informasi resources dari *server backend* 2. *Server backend* 3 bertugas sebagai *web server* 3 untuk menyajikan data atau

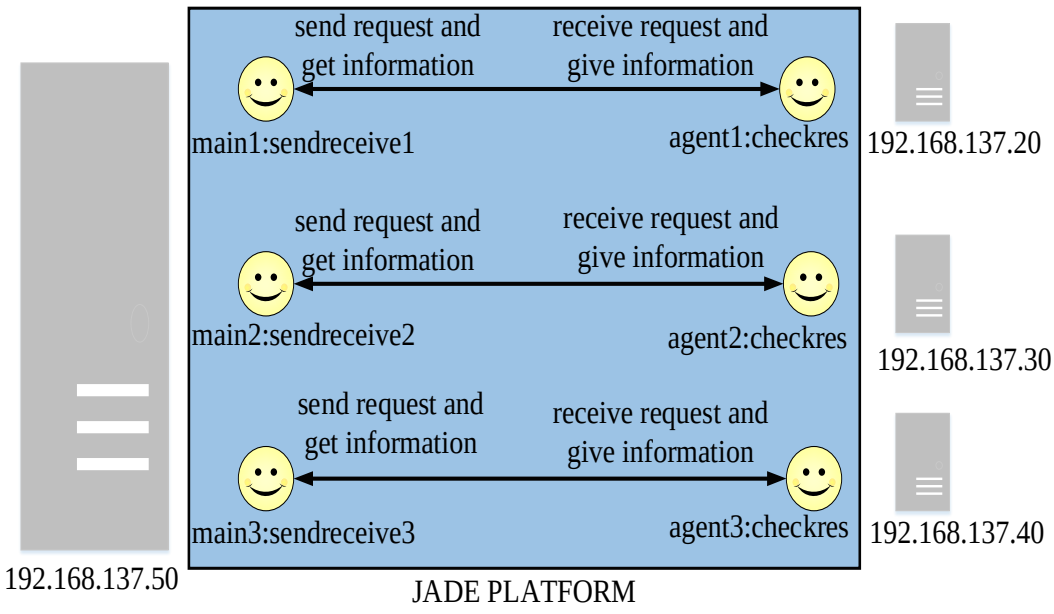
informasi kepada pengguna dan sebagai *agent* mengambil informasi resources dari *server backend* 3. *Server* pengguna bertugas sebagai *server* yang bertindak seperti pengguna untuk mendapatkan data atau informasi serta sebagai *benchmark* kemampuan dari sistem yang telah dibuat. Sistem operasi yang digunakan pada arsitektur ini adalah Debian 8 Server dan Windows Server 2012 standard.

Hypervisor yang digunakan pada PROXMOX adalah QEMU dengan *KVM Module*. *Hypervisor* ini berfungsi sebagai penghubung antara mesin virtual dan mesin fisik. PROXMOX berfungsi sebagai aplikasi pembuat mesin virtual. Mesin virtual dalam arsitektur dimaksudkan untuk menjalankan beberapa proses sekaligus yang terbagi dalam beberapa mesin *virtual*, karena mesin fisik hanya dapat menjalankan satu proses saja. Mesin *virtual* dapat mengefisienkan waktu pengerjaan, biaya, dan ruang karena kemudahan dan manfaat yang diberikan oleh mesin *virtual*.

D. Arsitektur Multi Agent System

Konfigurasi *multi agent system* dilakukan dengan memanfaatkan aplikasi JADE yang berbasis JAVA. *Agent* yang digunakan pada *multi agent system* ini terbagi menjadi dua kelompok fungsi. Kelompok fungsi yang pertama adalah *agent* yang berfungsi meminta informasi *resources* dari *server backend*. Kelompok fungsi yang kedua adalah *agent* yang berfungsi mengambil dan memberikan informasi *resources* dari *server backend*. Setiap kelompok fungsi terdiri atas tiga *agent*, sehingga total *agent* secara keseluruhan adalah enam *agent*. Gbr. 3 menjelaskan komunikasi dan relasi antar *agent* pada sistem yang dibuat.

Gbr. 3 menjelaskan bahwa arsitektur pada *multi agent system* memakai enam *agent* secara keseluruhan. Setiap *agent* pada sistem memiliki properti sendiri-sendiri. *Agent* 1 berada pada *server load balancing* dengan nama main1, yang bertugas untuk melakukan *request* informasi *resources* dan *get* informasi *resources* dari *server backend* 1. Kelas JAVA yang digunakan pada *agent* 1 adalah sendreceive1 dan jenis *behaviour TickerBehaviour*.*Port* yang digunakan pada *agent* 1 adalah 1090, dengan *protocol* transport adalah `http://192.168.137.50:7778/acc`. *Agent* 2 berada pada *server load balancing* dengan nama main2, yang bertugas untuk melakukan *request* informasi *resources* dan *get* informasi *resources* dari *server backend* 2. Kelas JAVA yang digunakan pada *agent* 2 adalah sendreceive2 dan jenis *behaviour TickerBehaviour*.*Port* yang digunakan pada *agent* 2 adalah 1091, dengan *protocol* transport adalah `http://192.168.137.50:7779/acc`. *Agent* 3 berada pada *server load balancing* dengan nama main3 yang bertugas untuk melakukan *request* informasi *resources* dan *get* informasi *resources* dari *server backend* 3. Kelas JAVA yang digunakan pada *agent* 3 adalah sendreceive3 dan jenis *behaviour TickerBehaviour*.*Port* yang digunakan pada *agent* 3 adalah 1092, dengan *protocol* transport adalah `http://192.168.137.50:7780/acc`. *Agent* 4 berada pada *server backend* 1 dengan nama *agent*1 yang bertugas untuk menunggu informasi *request* dan memberikan informasi *resources server backend* 1. Kelas JAVA yang digunakan pada *agent* 4 adalah checkres dan jenis *behaviour CyclicBehaviour*. *Port* yang digunakan pada *agent* 4 adalah 1099, dengan *protocol* transport adalah `http://192.168.137.20:7778/acc`. *Agent* 5 berada pada *server backend* 2 dengan nama *agent*2 yang bertugas untuk menunggu informasi *request* dan memberikan informasi *resources server backend* 2. Kelas JAVA yang digunakan pada *agent* 5 adalah checkres dan jenis *behaviour CyclicBehaviour*. *Port* yang digunakan pada *agent* 5 adalah 1099, dengan *protocol* transport adalah `http://192.168.137.30:7778/acc`. *Agent* 6 berada pada *server backend* 3 dengan nama *agent*3 yang bertugas untuk menunggu informasi *request* dan memberikan informasi *resources server backend* 3. Kelas JAVA yang digunakan pada *agent* 6 adalah checkres dan jenis *behaviour CyclicBehaviour*. *Port* yang digunakan pada *agent* 6 adalah 1099, dengan *protocol* transport adalah `http://192.168.137.40:7778/acc`.



Gbr. 3 Arsitektur *multi agent system.*

Message Transport Protocol (MTP) merupakan standar protokol-protokol pengiriman pesan yang didefinisikan oleh FIPA untuk dapat saling berkomunikasi antar platform. Jenis *behaviour* JADE pada *multi agent system* hanya terdiri atas dua jenis, yaitu *TickerBehaviour* dan *CyclicBehaviour*. *TickerBehaviour* merupakan jenis *behaviour* pada JADE yang memungkinkan pengguna/*programmer* untuk membuat *agent* melakukan tugas berulang kali selama hidupnya (*looping*) dengan menentukan waktu jeda untuk *agent* tersebut digunakan. *CyclicBehaviour* merupakan jenis *behaviour* JADE yang memungkinkan pengguna/*programmer* untuk membuat *agent* melakukan tugasnya dengan menunggu pesan ACL yang masuk dari *agent* lain dan melaksanakan tugasnya setelah pesan ACL tersebut diterima.

E. Parameter Pengujian

Pada setiap pengujian terdapat beberapa parameter secara spesifik yang harus dipenuhi. Setiap parameter tersebut dijelaskan pada bagian ini. Parameter-parameter diajukan berdasarkan *resources* mesin *server* dan perubahan nilai beban dari setiap *server backend*, sesuai dengan banyaknya *agent* dan *request* yang masuk. Parameter–parameter yang diajukan adalah sebagai berikut.

1) Jumlah Host Server: Jumlah maksimal *node server backend* pada penelitian yang dilakukan adalah tiga *node server backend*. Setiap *node server backend* dapat saling berkomunikasi melalui jaringan dan berisi konten *standard CMS Wordpress*. Komunikasi jaringan dilakukan melalui jaringan virtual pada perangkat lunak PROXMOX. Terdapat satu node *load balancing server* dan juga satu node *server* pengguna sebagai penguji kinerja sistem. Secara keseluruhan terdapat lima *node server*.

2) Jumlah Request: Jumlah *request* pengguna/pengguna pada penelitian ini adalah sebanyak 600 dan 1.800 *request*. Percobaan pertama dilakukan dengan 600 kali *request* selama 10s secara simultan menggunakan Apache JMeter. Percobaan kedua dilakukan dengan 1.800 kali *request* selama 10s secara simultan menggunakan Apache JMeter. Aplikasi akan membangkitkan sejumlah *request* langsung yang langsung masuk ke *server load balancing*.

3) Algoritme dan Agent: Algoritme menjadi tolok ukur sistem *load balancing* berjalan dengan semestinya atau tidak pada penelitian yang diajukan. Algoritme yang digunakan adalah LFB. Jumlah *agent* yang digunakan terdiri atas tiga *agent* ke semua *server* secara penjadwalan (*scheduling*). Secara total ada enam *agent* yang digunakan pada penelitian ini. Hal ini dimaksudkan agar tidak terjadi *bottleneck* pada jaringan dan meningkatkan waktu respons sistem.

F. Metode Test

Metode *test* merupakan suatu cara atau metode untuk menguji sebuah penelitian yang dilakukan dengan parameter-parameter sistem. Metode *test* dilakukan untuk memperoleh hasil atau informasi secara kuantitatif tentang sistem yang diujikan. Hasil yang diperoleh dari *test* ini merupakan rata-rata waktu respons, *throughput*, jumlah *error*, dan nilai deviasi dari sistem *load balancing*. Metode *test* diambil berdasarkan metode tes pada penelitian sebelumnya, yaitu menggunakan algoritme WLC untuk melakukan *load balancing* pada sepuluh *server* virtual dengan *resources server* yang sama. Jumlah *request* yang digunakan pada pengujian adalah 800 *request* [16]. Algoritme WLC ini merupakan gabungan antara algoritme *weight* dan *least connection*. *Request* tersebut diolah secara merata oleh algoritme WLC ini dengan nilai konfigurasi *weight* yang telah ditentukan sesuai *request* per *server*. Durasi yang digunakan untuk melakukan pengujian pada penelitian ini adalah 10s. Algoritme WLC ini dapat menjaga keseimbangan antar *server* dan mendapatkan waktu respons yang singkat. Metode *test* dari penelitian lainnya menggunakan *mobile agent* untuk melakukan *load balancing server* [17]. Konsep yang digunakan pada penelitian ini adalah dengan melakukan pemindahan beberapa proses dari *node* yang *overload* ke *node* yang normal. Algoritme *load balancing* yang digunakan adalah dengan menentukan *node overload* atau normal (*underloaded*). *Agent* bertugas memonitor semua informasi *load* setiap *server*. Pengujian penelitian ini menggunakan aplikasi JAVA J2SDK 1.6. Tugas yang masuk diberikan secara acak dan dikirim ke setiap *node server*. Jangka waktu kedatangan tugas yang masuk ke setiap *node server* rata-rata 0,1s. Waktu pemrosesan untuk tugas yang masuk secara distribusi antara 1s dan 10.000s. Untuk setiap simulasi yang dijalankan, tugas yang berhasil diberikan dan diproses adalah 500 tugas. Metode *test* terakhir yang dijadikan referensi adalah metode *test* yang menggunakan koneksi maksimal yang berasal dari *request* pengguna/pengguna untuk mengakses *server* [1]. Koneksi maksimal ini merupakan koneksi terbanyak yang dapat ditangani oleh *server*. *Agent* mengunjungi setiap *server* dan memilih *server* yang memiliki *load* rendah sebagai *server* terbaik. *Agent* menghentikan sementara *(pause) server* dengan *load server* di bawah *threshold* dan memilihnya sebagai *server* terbaik dan dipilih untuk menggantikan *server* yang *overload*.

Berdasarkan ketiga metode *test*, pada penelitian yang dilakukan dipilih metode *test* yang pertama. Metode *test* pertama dipilih karena lingkungan penelitiannya hampir sama dengan lingkungan penelitian yang dilakukan. Metode *test* penelitian sistem *load balancing* terdistribusi menggunakan metode LFB-MAS ini menggunakan algoritme *load balancing* yang dijelaskan sebelumnya untuk mencari nilai waktu respons, *throughput*, *error*, dan deviasi. Metode LFB-MAS juga digunakan untuk mempertahankan *request* yang banyak dari pengguna yang masuk ke sistem. *Agent* diatur secara *scheduling*/penjadwalan untuk mengambil *resources* masing-masing *server*. Metode LFB-MAS ini dibandingkan dengan algoritme WLC dengan metode *test* menggunakan 1.800 *request* selama 10s. Pada metode *test* yang kedua dilakukan perbandingan antara LFB-MAS dengan algoritme LFB tanpa menggunakan *agent* dengan 1.800 *request* selama 10s. Aplikasi yang digunakan dalam pengujian adalah Apache JMeter. Pada metode ini juga dilakukan perbandingan dengan algoritme LFB tanpa menggunakan *agent*. Hasil yang diperoleh akan dianalisis kemudian diambil sebuah kesimpulan.

Pada skema ini, pada setiap *server backend* tertanam *agent* yang siap untuk memberikan informasi *resources* dari setiap *server backend*. *Server* utama menjadi pusat perkumpulan

informasi setiap *server backend*. *Server* utama membuat tabel informasi yang berisi informasi *resources* dari setiap *server backend*. Informasi tersebut nantinya digabungkan dengan algoritme *load balancing*. Informasi *resources* yang dikumpulkan adalah beban CPU dan beban memori. Banyaknya koneksi yang dapat diproses berbanding lurus dengan *load* CPU dan memori. Berdasarkan hal tersebut, diperoleh rumus yang digunakan untuk mengukur *load* sebuah *server* sebagai berikut.

$$load = \frac{\%CPU.\%memory}{N_{request}} . \quad (1)$$

Persamaan (1) merupakan persamaan yang diperoleh berdasarkan persentase beban CPU ($\%CPU$) dikalikan dengan persentase beban memori ($\%memory$.) Nilai dari perkalian tersebut dibagi oleh jumlah *request* yang masuk. Nilai persentase beban CPU dan beban memori dijadikan sebagai patokan bahwa *server backend* tergolong *overload* atau tidak *overload*. Jika nilai *load* kurang dari atau sama dengan nilai *threshold*, maka *server* tersebut dapat dikatakan normal atau tidak *overload*. Sebaliknya, jika nilai melebihi nilai *threshold*, maka *server* tersebut dapat dikatakan sibuk atau *overload*. Nilai *threshold* yang diajukan adalah 90%. Setelah menentukan nilai *load* dari setiap *server backend*, nilai digabungkan dengan algoritme *load balancing* LFB. *Server* dengan nilai *load* yang tinggi atau melebihi *threshold* ditunda sementara dahulu dari kelompok *load balancing* sampai *load server* tersebut menurun dan normal kembali.

Waktu respons merupakan nilai hasil yang menjadi data dari penelitian ini. Nilai *respons time* diperoleh dari waktu respons *server* kepada pengguna. Nilai waktu respons dapat dijadikan sebuah variabel yang dimasukkan ke dalam persamaan-persamaan. Persamaan nilai rata rata dapat diketahui dengan menggunakan variabel waktu respons. Persamaan (2) merupakan persamaan pencarian nilai rata rata waktu respons.

$$Average = \frac{1}{n_{request}} \sum_{i=1}^{n} t_i. \quad (2)$$

Variabel n merupakan banyaknya *request* yang diuji. Variabel t merupakan nilai dari waktu respons setiap *request*. Nilai rata rata diperoleh dari pembagian antara jumlah waktu respons dengan banyaknya *request*, mengacu pada (2). Persaman-persamaan lain dapat dihasilkan dengan menggunakan variabel waktu respons dan jumlah *request* dari pengguna. Persamaan median, varian, dan standar deviasi merupakan persamaan lain yang dapat digunakan.

$$Median = \frac{1}{2}\left(t\ \ \frac{n_{request}}{2}\ \ + t\left(\ \frac{n_{request}}{2}\ \ + 1\right)\right) \quad (3)$$

$$Variance = \frac{1}{n} \sum_{i=1}^{n} (t - \bar{t})^2 \quad (4)$$

$$Std.Deviation = \sqrt{variance} \quad (5)$$

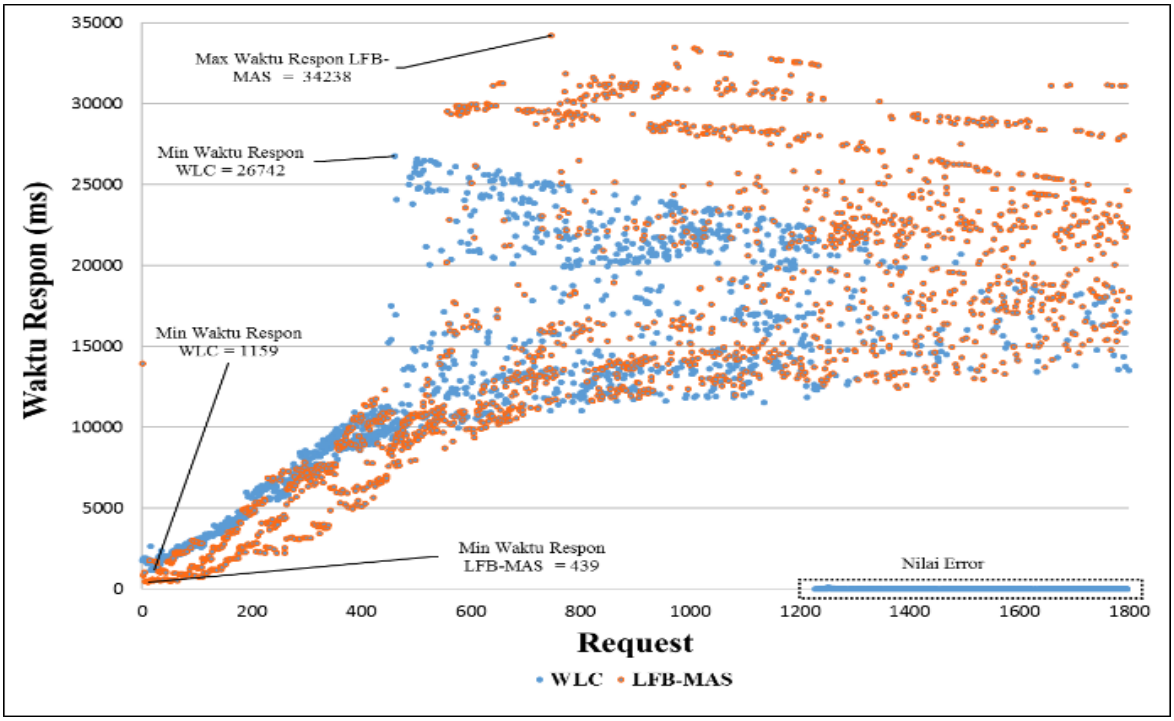
$$Throughput = \frac{n}{((x_{tmax} + t_{max}) - x_{tmin})} \quad (6)$$

$$Error = \frac{n_{error}}{\mathrm{n}_{request}}\ x\ 100\% \quad (7)$$

Nilai median atau nilai tengah diketahui dengan menggunakan (3). Nilai median diketahui dengan nilai waktu respons (t) ke banyaknya *request* (n) dibagi dua ditambah dengan nilai waktu respons setelahnya. Hasil dari penjumlahan nilai tersebut dibagi dua. Nilai varian dan standar deviasi dicari menggunakan (4) dan (5). Nilai varian yang ditunjukkan pada (4) adalah jumlah kuadrat dari selisih nilai waktu respons (t) dari nilai rata-ratanya, kemudian dibagi dengan jumlah nilai *request* (n). Nilai varian digunakan untuk mengetahui seberapa jauh persebaran nilai waktu respons terhadap rata-rata. Persamaan (5) merupakan suatu nilai yang menunjukkan tingkat variasi suatu kelompok data. Nilai standar deviasi didapatkan dari akar nilai varian sebelumnya. Kelompok *server backend* terdiri atas tiga *server* seperti yang telah dijelaskan sebelumnya. Nilai *throughput* dari masing masing *server* pada (6) diperoleh dari banyaknya *request* yang masuk dibagi dengan cap waktu (x) pada t max ditambah dengan t max kemudian dikurangi dengan cap waktu (x) pada t min sesuai hasil pengujian. *Throughput* merupakan salah satu komponen dari kebutuhan nonfungsional yang berada di dalam kategori kinerja dan diukur sebagai total nomor transaksi atau permintaan dalam waktu tertentu atau transaksi per detik (TPS). Ini cara untuk menunjukkan kapasitas *server* dalam hal banyak beban yang dapat diambil atau ditangani. Dalam perspektif jaringan, *throughput* juga bisa berarti banyaknya jumlah *byte* yang ditransfer atau dikirim per detik (B/detik atau kB/sec). Adapun nilai *error* yang dihasilkan pada pengujian yang dilakukan dengan aplikasi *benchmark* Apache JMeter yaitu nilai pada variabel *connect time* yang melebihi 2.100 ms dianggap *error*. Nilai *error* ini merupakan *request* yang gagal atau tidak diproses pada *server* karena melebihi kapasitas dan waktu yang ditentukan. Nilai *error* pada (7) didapatkan dari informasi gagal oleh perangkat lunak yang menandakan *connection time out* atau *connection reset* dikali 100%. Informasi gagal atau *error* biasanya dikarenakan oleh *server* yang tidak dapat menampung proses yang melebihi batas dari sistem. Berikut contoh *error* yang melebihi batas dari kemampuan *server* pada *apache* = " [*mpm_prefork:error] [pid 1917] AH00161: server reached MaxRequestWorkers setting, consider raising the MaxRequestWorkers setting*". Berdasarkan *error* tersebut, sistem harus dinaikkan proses *child* pada *server*. Informasi tentang *load server* dicatat pada *server* utama sebagai log informasi. Log informasi ini untuk mempermudah apabila ada kendala atau *error* di salah satu *server backend* atau *agent*.

IV. ANALISIS KINERJA

Bagian ini memaparkan hasil pengujian dan analisis penelitian yang telah dilakukan dengan metode *load balancing* LFB yang dikombinasikan dengan *multi agent system* (LFB-MAS). Metode LFB-MAS akan diuji kinerjanya dalam melakukan pemrosesan data. Analisis pengujian terbagi menjadi dua bagian, yaitu pengujian metode WLC dengan LFB-MAS dan pengujian LFB-MAS tanpa *agent*. Analisis pengujian dilakukan berdasarkan hasil yang diperoleh pada saat pengujian sistem *load balancing*. Data pengujian disajikan dalam bentuk grafik dan berupa data tabel. Pengujian sistem dilakukan dengan memberikan 1.800 *request* ke dalam sistem *load balancing* selama 10s. Analisis pengujian ini diharapkan dapat memberikan gambaran kinerja dari setiap metode yang diuji.



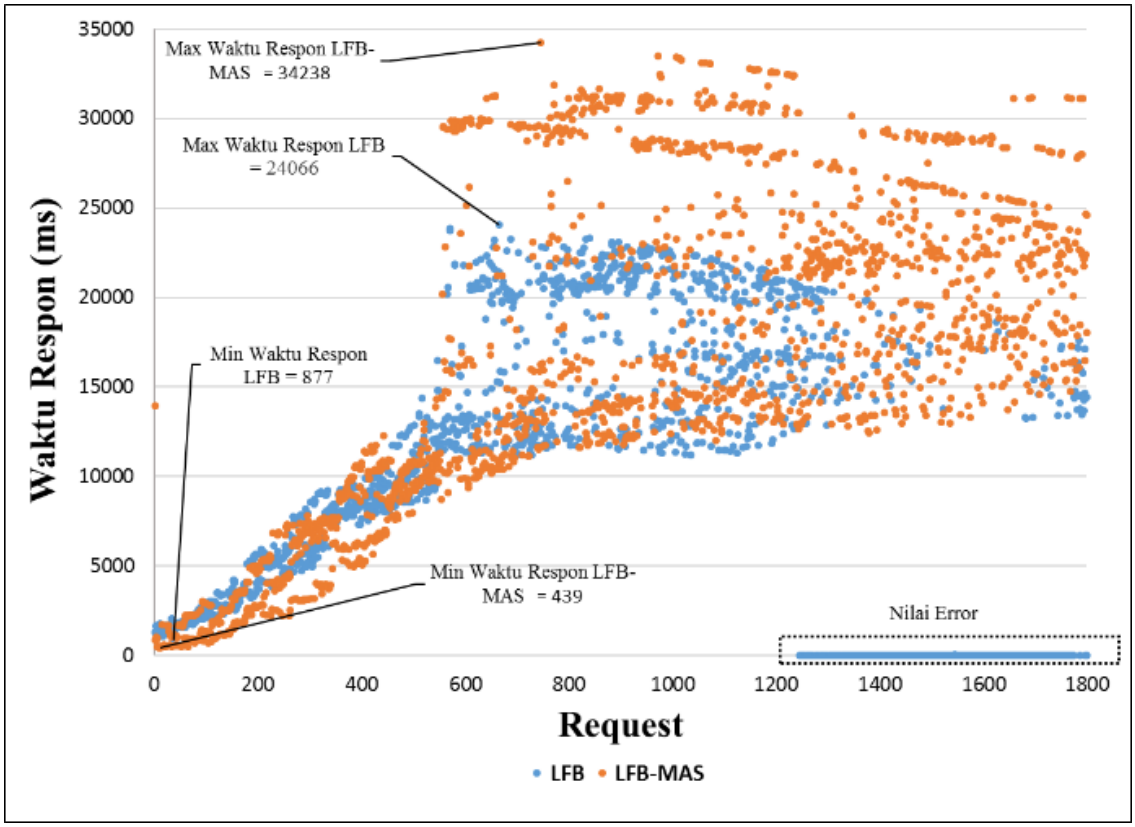
Gbr. 4 Perbandingan waktu respon metode LFB-MAS dengan WLC

Gbr. 4 menjelaskan perbandingan waktu respons metode WLC dengan LFB terhadap 1.800 *request* selama 10s. Waktu respons yang dihasilkan metode WLC kurang dari 30.000 ms karena waktu respons maksimalnya adalah 26.742 ms. Waktu respons yang dihasilkan metode LFB-MAS lebih besar dibandingkan dengan metode LFB-MAS, yaitu sebesar kurang dari 35.000 ms karena waktu respons maksimalnya adalah 34.238 ms. Akan tetapi, metode LFB lebih tangguh dalam menangani sejumlah *request* yang besar. Tabel II menjelaskan secara rinci hasil pengujian kedua tahap kedua ini. Berdasarkan Tabel II, metode WLC hanya mampu menahan sekitar 74,50% dari 1.800 *request*. Metode LFB-MAS dapat menahan 100% dari 1.800 *request* meskipun kalah pada hal waktu respons, *throughput*, dan lain lain. Nilai *error* yang ditampilkan pada grafik merupakan banyaknya *request* yang tidak dapat diproses oleh *server*. Nilai *error* ini didapatkan karena batas kemampuan dari *server* ketika pemrosesan *request* sehingga *request* yang selanjutnya akan ditolak. Nilai *error* ini dijelaskan dengan rinci pada metode *test* pada bagian sebelumnya. Berdasarkan pengujian kedua ini diperleh hasil bahwa metode LFB-MAS lebih kuat dan andal dibandingkan dengan metode WLC.

A. Pengujian LFB-MAS dan Tanpa Agent

Hasil dari penelitian sebelumnya menunjukkan nilai *throughput* dan nilai waktu respons yang kurang baik. Hasil dari pengujian ini merupakan hasil perbandingan *load balancing* menggunakan metode LFB-MAS dan algoritme LFB tanpa menggunakan *agent*. Parameter pengujian masih sama dengan penelitian sebelumnya, yaitu dengan menggunakan 1.800 *request* pada masing masing sistem *load balancing* selama 10s. Hasil dari pengujian pada penelitian ini berupa grafik dan data tabel.

Gbr. 5 merupakan grafik perbandingan yang menggambarkan waktu respons metode *load balancing* dengan algoritme LFB tanpa dikombinasikan dengan *agent* dan metode LFB-MAS. Metode LFB-MAS terlihat lebih stabil dalam menjaga dan memproses 1.800 *request*. Metode LFB tanpa menggunakan *multi agent system* terlihat tidak dapat mempertahankan 1.800 *request* setelah melewati 1.200 *request*. Metode LFB-MAS dapat memproses 1.800 *request* dengan waktu respons dibawah 3.500 ms. Tabel III merupakan data hasil pengujian metode LFB tanpa *agent* dan LFB-MAS. Berdasarkan data yang diperoleh dari Tabel III, LFB tanpa menggunakan *agent* hanya mampu memproses 75,61% dari 1.800 *request*. LFB tanpa menggunakan *multi agent system* hanya dapat memproses 1.361 *request*. Metode LFB-MAS dapat memproses semua *request*. Nilai *error* yang ditampilkan pada grafik merupakan banyaknya *request* yang tidak dapat diproses oleh *server*. Nilai *error* ini diperoleh karena batas kemampuan dari *server* ketika pemrosesan *request*, sehingga *request* yang selanjutnya akan ditolak. Nilai *error* ini dijelaskan dengan rinci pada metode *test* pada bagian sebelumnya. Nilai waktu respons dan *throughput* dari metode LFB-MAS juga tidak begitu jauh selisihnya, yaitu 16.190 ms berbanding 12.691 ms dan 43 *request*/s berbanding 47,4 *request*/s. Berdasarkan hasil tersebut, dapat dibuktikan bahwa sistem *load balancing* menggunakan metode LFB-MAS lebih andal dibandingkan tanpa menggunakan *multi agent system*. Berdasarkan hasil pada data percobaan yang telah diperoleh, LFB-MAS memang menghasilkan waktu respons yang lebih lama karena memproses lebih banyak *request* yang dapat diproses menunggu sampai pemrosesan *request* tersebut selesai, sedangkan LFB tanpa *agent* dan WLC mendapatkan waktu *respons* lebih cepat dikarenakan *request* yang banyak tidak dapat diproses sehingga pemrosesan *request* langsung selesai.



Gbr. 5 Perbandingan waktu respons metode LFB tanpa *agent* dan LFB-MAS.

TABEL II
DATA PENGUJIAN METODE LFB-MAS DAN WLC DENGAN 1.800 *REQUEST*

Metode	*Request*	*Average* (ms)	Std. Dev. (ms)	*Error* %	*Throughput* (*request*/s)
WLC	1.800	13.614	6.893,10	25,50	45,3
LFB-MAS	1.800	16.190	9.170,54	0,00	43,0

TABEL III
DATA PENGUJIAN METODE LFB TANPA *AGENT* DAN LFB-MAS DENGAN 1.800 *REQUEST*.

Metode	*Request*	*Average* (ms)	Std. Dev. (ms)	*Error* %	*Throughput* (*request*/s)
LFB Tanpa *Agent*	1.800	12.691	6.504,89	24,39	47,4
LFB-MAS	1.800	16.190	9.170,54	0,00	43,0

Hasil dari analisis pengujian ini membuktikan bahwa sistem *load balancing* dapat ditingkatkan kinerjanya dengan menggunakan metode LFB-MAS dan metode LFB-MAS dengan menggunakan *multi agent system* dapat dijadikan sebagai referensi *load balancing* dalam menangani sejumlah proses dengan skala besar yang efisien, andal, dan mengurangi risiko *server* menjadi *overload*.

V. KESIMPULAN

Berdasarkan hasil pengujian dan analisis pada Bagian IV, dapat diambil kesimpulan bahwa rancang bangun sistem *load balancing* menggunakan algoritme LFB-MAS berhasil membagi beban secara merata ke semua *server* pada skala besar dan lebih baik dari penelitian sebelumnya. Algoritme LFB merupakan sebuah algoritme *least connection* yang dikombinasikan dengan melakukan pengecekan waktu respons pada *byte* pertama pada data dan waktu respons yang paling cepat yang menjadi prioritas. *Multi agent system* memungkinkan sistem *load balancing* untuk dapat menjaga kestabilan dan keandalan. Algoritme LFB-MAS mendapatkan hasil yang lebih baik dibandingkan metode WLC, dengan waktu respons 16.190, *error* 0,00%, dan nilai *throughput* sebesar 43,0 *request*/s. Dari hasil yang telah diperoleh, LFB-MAS memang menghasilkan waktu respons yang lebih lama dikarenakan LFB-MAS memproses lebih banyak *request* yang dapat diproses menunggu sampai pemrosesan *request* tersebut selesai, sedangkan LFB tanpa *agent* dan WLC mendapatkan waktu respons lebih singkat karena *request* yang banyak tidak dapat diproses, sehingga pemrosesan *request* langsung selesai.

REFERENSI

[1] J. Cao, Y. Sun, X. Wang, and S. K. Das, "Scalable load balancing on distributed web servers using *mobile agents*," *J. Parallel Distrib. Comput.*, vol. 63, no. 10, pp. 996–1005, 2003.
[2] Q. Long, J. Lin, and Z. Sun, "Agent scheduling model for adaptive dynamic load balancing in agent-based distributed simulations," *Simul. Model. Pract. Theory*, vol. 19, no. 4, pp. 1021–1034, 2011.
[3] M. A. Metawei, S. A. Ghoneim, S. M. Haggag, and S. M. Nassar, "Load balancing in distributed multi-agent computing systems," *Ain Shams Eng. J.*, vol. 3, no. 3, pp. 237–249, 2012.
[4] A. Yaseen, H. Ji, and Y. Li, "A load-balancing workload distribution scheme for three-body interaction computation on Graphics Processing

Units (GPU)," *J. Parallel Distrib. Comput.*, vol. 87, pp. 91–101, 2016.
[5] O. Rihawi, Y. Secq, and P. Mathieu, "Load-Balancing for Large Scale Situated Agent-Based Simulations," *Procedia - Procedia Comput. Sci.*, vol. 51, pp. 90–99, 2015.
[6] A. Singh, D. Junejab, and M. Malhotraa, "Autonomous Agent Based Load Balancing Algorithm in Cloud Computing," *Int. Conf. Adv. Comput. Technol. Appl. (ICACTA- 2015) Procedia Computer Science*, vol. 45, pp. 832–841, 2015.
[7] P. Braun and W. Rossak, *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*. Massachusetts, United States: Morgan Kaufmann, 2004.
[8] V. Baousis, S. Hadjiefthymiades, G. Alyfantis, and L. Merakos, "Autonomous *mobile agent* routing for efficient server resource allocation," *J. Syst. Softw.*, vol. 82, no. 5, pp. 891–906, 2009.
[9] X.-J. Shen *et al*., "Achieving dynamic load balancing through *mobile agent*s in small world P2P networks," *Comput. Networks*, vol. 75, pp. 134–148, 2014.
[10] F. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. Chichester, England John Wiley & Sons Ltd, 2007.
[11] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "JADE: A software framework for developing multi-agent applications. Lessons learned," *Inf. Softw. Technol.*, vol. 50, no. 1–2, pp. 10–21, 2008.
[12] A. V. Sandita and C. I. Popirlan, "Developing A Multi-Agent System in JADE for Information Management in Educational Competence Domains," *Procedia Econ. Financ.*, vol. 23, no. October 2014, pp. 478–486, 2015.
[13] C. V. Trappey, A. J. C. Trappey, C. J. Huang, and C. C. Ku, "The design of a JADE-based autonomous workflow management system for collaborative SoC design," *Expert Syst. Appl.*, vol. 36, no. 2 PART 2, pp. 2659–2669, 2009.
[14] R. Z. Khan and J. Ali, "Classification of Task Partitioning and Load Balancing Strategies in Distributed Parallel Computing Systems," *Int. J. Comput. Appl.*, vol. 60, no. 17, pp. 48–53, 2012.
[15] Y. Jiang, "A Survey of Task Allocation and Load Balancing in Distributed Systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9219, no. c, pp. 1–1, 2015.
[16] D. Choi, K. S. Chung, and J. Shon, "An Improvement on the Weighted Least-Connection Scheduling Algorithm for Load Balancing in Web *Cluster* Systems," *T.-h. Kim al. GDC/CA 2010*, vol. CCIS 121, pp. 127–134, 2010.
[17] N. Eftekhari, F. H. Zeinalabedin, and A. T. Haghighat, "A Novel Threshold-Based Dynamic Load Balancing Algorithm Using Mobile Agent in Distributed System," *V.V. Das N. Thankachan CIIT 2011*, vol. CCIS 250, pp. 103–109, 2011.
[18] R. B. Patel and N. Aggarwal, "Load balancing on open networks: a *mobile agent* approach," *J. Comput. Sci.*, vol. 2, no. 4, pp. 337–346, 2006.