TESIS – KI142501

OPTIMASI DAYA DATA CENTER CLOUD COMPUTING PADA WORKLOAD HIGH PERFORMANCE COMPUTING (HPC) DENGAN SCHEDULING PREDIKTIF SECARA REAL TIME

Amirullah
5113201011

DOSEN PEMBIMBING
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
Hudan Studiawan, S.Kom., M.Kom.

PROGRAM MAGISTER
BIDANG KEAHLIAN KOMPUTASI BERBASIS JARINGAN
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2016

THESIS – KI142501

POWER OPTIMIZATION OF CLOUD COMPUTING DATA CENTER ON HIGH PERFORMANCE COMPUTING (HPC) WORKLOAD WITH PREDICTIVE IN REAL TIME SCHEDULLING

Amirullah
5113201011

SUPERVISOR
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.
Hudan Studiawan, S.Kom., M.Kom.

MASTER PROGRAM
AREAS OF EXPERTISE NETWORK-BASED COMPUTING
INFORMATICS ENGINEERING
FACULTY OF INFORMATION TECHNOLOGY
INSTITUTE OF TECHNOLOGY SEPULUH NOPEMBER
SURABAYA
2016

Tesis ini disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M. Kom)
di
Institut Teknologi Sepuluh Nopember

oleh:
Amirullah
Nrp. 5113201011

Tanggal Ujian: 29 Juni 2016
Periode Wisuda: September 2016

Disetujui oleh:

1. Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D. ..... (Pembimbing I)
   NIP. 19770824 200604 1 001
2. Hudan Studiawan, S.Kom., M.Kom. ..... (Pembimbing II)
   NIP. 19870511 201212 1 003
3. Waskitho Wibisono, S.Kom., M.Eng., PhD. ..... (Penguji I)
   NIP. 19741022 200003 1 001
4. Tohari Ahmad, S.Kom., MIT., Ph.D. ..... (Penguji II)
   NIP. 19750525 200312 1 002
5. Dr. Eng. Radityo Anggoro, S.Kom., M.Sc. ..... (Penguji III)
   NIP. 19841016 200812 1 002

Direktur Program Pascasarjana,

Prof.Ir. Djauhar Manfaat, M.Sc, Ph.D
NIP. 19601202 198701 1 001

OPTIMASI DAYA DATA CENTER CLOUD COMPUTING PADA WORKLOAD HIGH PERFORMANCE COMPUTING (HPC) DENGAN SCHEDULING PREDIKTIF SECARA REAL TIME

Nama mahasiswa	: Amirullah
NRP	: 5113201011
Pembimbing I	: Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D
Pembimbing II	: Hudan Studiawan, S.Kom, M.Kom

ABSTRAK

Tantangan terbesar yang muncul pada data *center cloud computing* adalah meningkatnya biaya konsumsi daya. Pengembangan data *center* akan bertolak belakang dengan penghematan daya, semakin tinggi performa sebuah data *center*, maka semakin tinggi pula konsumsi energi yang dibutuhkan, hal ini disebabkan oleh kebutuhan jumlah server ataupun *hardware* pada data *center* yang semakin meningkat.

Data *center cloud computing* yang berbasis *High Performance Computing* (HPC) merupakan sebuah teknologi yang dibangun dari kumpulan server dalam jumlah besar untuk menjamin ketersediaan tinggi dari sebuah *cloud computing*, namun sebenarnya beberapa server tersebut hanya direncanakan untuk beban puncak yang jarang atau tidak pernah terjadi. Ketika beban pada titik terendah, maka server tersebut akan berada dalam kondisi *idle*.

Optimasi daya menggunakan DNS (*Dynamics Shutdown*) dengan memanfaatkan kondisi beban rendah server dapat menjadi solusi yang tepat untuk mengurangi konsumsi daya pada data *center*. Namun jika optimasi tersebut dilakukan dengan konvensional dan hanya berdasarkan data *real time*, maka kemungkinan besar akan berpengaruh terhadap performa data *center*.

Optimasi yang dilakukan pada penelitian ini adalah dengan metode prediksi menggunakan moving *average* untuk menentukan penjadwalan DNS. Hasil pengujian dengan komputer virtual menunjukkan bahwa dengan metode prediksi dapat mengurangi konsumsi daya sebesar 1,14 Watt dibandingkan dengan metode konvensional.

Kata kunci: *Cloud computing*, *Data Center*, Optimasi Daya, *Dynamics Shutdown*, Prediksi, *Real Time*, *moving average*, penjadwalan

POWER OPTIMIZATION OF CLOUD COMPUTING DATA CENTER ON HIGH PERFORMANCE COMPUTING (HPC) WORKLOAD WITH PREDICTIVE IN REAL TIME SCHEDULLING

Nama mahasiswa : Amirullah
NRP : 5113201011
Pembimbing I : Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D
Pembimbing II : Hudan Studiawan, S.Kom, M.Kom

ABSTRACT

The biggest challenge that emerged in the data center cloud computing is increasing costs for power consumption. The higher performance of a data center, the higher the energy consumption required, this was due to the needs of the number of servers or hardware in the data centers are growing.

Data center cloud computing-based High Performance Computing (HPC) is a platform built on very large number of servers to ensure high availability of cloud services. The workload is likely to be happening at specific periods of time, instead of all time. Therefore, when the load is at its lowest point, the server will be in the idle condition.

Implementing DNS (Dynamics Shutdown) at appropriate times could be a solution for reducting power consumption in the data center. However, if the optimization is done only on the basis of conventional and real time data, it will most like-ly affect the performance of the data center.

In this study, optimization is done by conducting prediction using moving average method to determine the schedule of DNS. Results of testing with virtual computer shows that the prediction methods can reduce the power consumption of 1.14 Watts compared to conventional methods.

Keywords: Cloud computing, Data Center, Power optimization, Dynamics Shutdown, Prediction, Real Time, moving average, scheduling

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Pada beberapa tahun terakhir, perkembangan layanan *cloud computing* mengalami peningkatan yang sangat drastis, hal ini disebabkan karena adanya keterlibatan data *center* dan paradigma komputasi paralel. *Cloud computing* dan data *center* tidak dapat dipisahkan. Data *center* merupakan fasilitas untuk menempatkan sistem komputer dan komponen-komponen yang terkait (Sofana, 2012). Pengoperasian data *center* yang tersebar di wilayah yang luas memerlukan pertimbangan seberapa besar konsumsi energi terhadap total biaya pengoperasian dari data *center*.

Masalah utama dari infrastruktur *cloud* bukan hanya dari segi biaya yang mahal akan tetapi juga kurang ramah lingkungan. Biaya pemakaian energi yang tinggi kemudian emisi karbon yang dihasilkan akibat tingginya kebutuhan akan energi listrik baik untuk tujuan yang berhubungan dengan komputasi ataupun untuk tujuan pendukung operasional dari data *center*. Para penyedia layanan infrastruktur *cloud* perlu untuk mengukur agar *margin* keuntungan layanan *cloud* tidak tereduksi oleh tingginya biaya pemakaian energi listrik (Fathurahman et al., 2012).

Tantangan terbesar yang muncul pada data *center* adalah meningkatnya biaya konsumsi untuk daya. Seperti yang dijelaskan oleh Filani dalam penelitiannya, pada dekade terakhir biaya untuk daya dan pendingin data *center* telah meningkat sebesar 400% dan kecenderungannya akan terus meningkat. Pada beberapa kasus, konsumsi daya listrik memakan porsi 40-50% dari keseluruhan biaya operasional dari data *center* (Filani et al., 2008).

Pengembangan pada data *center* akan memiliki kebutuhan untuk menambah server baru agar dapat memenuhi lonjakan akses yang semakin tinggi. Namun berdasarkan survei terakhir oleh Filani, faktor penghambat utama dalam pengembangan data *center*, senilai 56% adalah berdasarkan konsumsi daya dan pendinginan (Filani et al., 2008). Dengan kondisi tersebut, maka data *center* saat ini menghadapi dua masalah besar yakni bagaimana mengembangkan layanan baru

tetapi dengan konsekuensi konsumsi daya untuk komputasi dan pendinginan terus meningkat. Jika kecenderungan ini terus terjadi, maka kemampuan data *center* untuk menambah layanan baru akan terhambat.

Untuk mengatasi hal ini, pengelola data *center* memiliki beberapa pilihan di antaranya sebagai berikut (Filani et al., 2008):

1. Menambah kapasitas daya dan pendingin
2. Membangun data *center* baru
3. Melakukan pengelolaan energi yang memaksimalkan kapasitas yang ada.

Dua pilihan awal akan sangat mahal karena melibatkan belanja modal dan instalasi baru. Maka pilihan ketigalah yang paling memungkinkan untuk mengatasi dua hal tersebut di atas.

Ada beberapa mekanisme optimasi daya yang dapat dilakukan untuk memaksimalkan kapasitas server komputasi yang ada, pertama adalah dengan mengurangi daya pada setiap server sesuai dengan kebutuhan dan yang kedua mematikan (*shutdown*) server yang dalam kondisi *idle* sehingga konsumsi energi bisa ditekan pada kondisi minimal.

Telah ada penelitian yang berfokus pada dua hal tersebut yaitu mengurangi daya dan mematikan (*shutdown*) daya pada server yang tidak diperlukan. Hasil dari beberapa penelitian menunjukkan bahwa penerapan dua mekanisme tersebut dapat menghasilkan tingkat optimasi yang sangat baik. Dalam penelitian (Choi et al., 2004) optimasi dengan cara mengurangi konsumsi daya pada CPU dilakukan dengan skema *Dynamic Voltage and Frequency Scaling* (DVFS) telah menunjukkan penghematan energi CPU hingga mencapai 80%. Penelitian (Kliazovich et al., 2012) telah mengkaji perbandingan optimasi daya antara DVFS dengan DVFS + DNS (*Dynamics Shutdown*) hasilnya menunjukkan bahwa penghematan yang lebih baik adalah dengan menggunakan DVFS + DNS.

DVFS dan DNS merupakan skema optimasi berdasarkan deteksi beban secara dinamis atau *real time* sesuai kondisi beban kerja pada server (Lee, 2009), artinya saat akses ataupun beban kerja pada server dalam keadaan rendah, sangat rendah atau bahkan tidak ada beban sama sekali, maka sistem akan mengurangi konsumsi daya pada server tersebut sehingga kunsumsi daya mendekati titik terendah. Ketika server berada pada kondisi *idle*, maka sistem akan melakukan

shutdown otomatis pada server tersebut sehingga konsumsi daya dapat diminimalkan.

Namun masalahnya adalah jika beban kembali melonjak secara tiba-tiba dan saat itu juga sistem membutuhkan tambahan komputasi dari server yang telah dikurangi kemampuannya atau telah di-*shutdown*, maka sistem akan membutuhkan waktu beberapa menit agar dapat menjadikan sistem kembali pada performa normal ataupun optimal. Semakin lama durasi waktu yang dibutuhkan, maka semakin lama pula performa server dapat dikembalikan, sehingga performa server secara keseluruhan juga akan ikut terganggu. Dan bagaimana jika ini terjadi pada ratusan bahkan ribuan server yang ada dalam data *center*, maka ini sangat berpengaruh pada kinerja keseluruhan sistem *cloud computing* yang telah dibangun.

Ada hal yang perlu diperhatikan untuk memaksimalkan skema tersebut, yaitu bagaimana melakukan penjadwalan yang baik sehingga proses DNS (*Dynamics Shutdown)* tersebut dapat dilakukan dengan tepat tanpa mengurangi performa pada kinerja data *center*. Penjadwalan yang kurang tepat dalam membaca atau mendeteksi fluktuasi akses pada setiap server dapat menyebabkan layanan data *center cloud* menjadi bermasalah. Akses yang sangat dinamis pada layanan *cloud* perlu diperhatikan secara detail untuk menangani penjadwalan optimasi. Dari masalah ini terlihat bahwa sistem tidak hanya dapat mengandalkan metode *real time*. Oleh karena itu perlu adanya nilai prediksi yang dilakukan sebelum sistem optimasi bekerja untuk menentukan kapan server tersebut benar-benar dapat dikurangi performanya dan kapan server dapat dimatikan untuk mengurangi konsumsi daya. Cara tersebut dapat dilakukan dengan mekanisme kombinasi antara metode prediksi dengan *real time.*

Penelitian mengenai penjadwalan pada optimasi daya telah dilakukan pada beberapa penelitian seperti (Lee, 2009) menggunakan metode penjadwalan *real time,* pada penelitian (Lakshmanan et al., 2010) menggunakan *scheduling parallel real time* sebagai metode penjadwalannya. Namun dalam beberapa penelitian tersebut belum membahas metode penjadwalan berdasarkan metode prediksi maupun kombinasi metode prediksi dengan *real time.* Penelitian mengenai metode prediksi atau prediktif pada *cloud computing* biasanya sering dilakukan pada penelitian *auto scaling*, seperti pada (Yang et al., 2013), (Roy et al., 2011) telah

membuat penelitian tentang *auto scaling* pada virtual untuk memprediksi *workload,* namun belum ada penelitian untuk penjadwalan optimasi daya menggunakan model prediksi dan kombinasi antara prediksi dan *real time*.

Dalam penelitian ini akan dilakukan metode optimasi daya pada server dengan skema *Dynamics Shutdown* (DNS) (Sueur and Heiser, 2010). Untuk penjadwalan akan digunakan kombinasi metode prediksi dan *real time*, sehingga proses optimasi dapat dilakukan dengan tepat tanpa mempengaruhi kinerja atau performa dari sistem data *center cloud computing*.

Metode penjadwalan pada penelitian ini akan digunakan pendekatan metode atau algoritma *moving average* yang kemudian akan dikombinasikan dengan *real time*. Metode *moving average* menggunakan nilai rataan sejumlah data masa lalu dengan menentukan jumlah periode tertentu untuk menentukan nilai prediksi yang akan datang (Yaffee and McGee, 2000). Penelitian metode *moving average* banyak digunakan untuk memprediksi fluktuasi saham seperti pada penelitian (Zulkarnain, 2012). Sebagaimana kita ketahui, saham memiliki fluktuasi yang sangat dinamis, namun dengan metode prediksi seperti penelitian (Zulkarnain, 2012) menunjukkan hasil yang sangat baik dalam mendeteksi perkembangan saham. Dalam penelitian *cloud computing* metode *moving average* digunakan sebagai metode prediksi pada *autoscaling* (Lorido-Botran et al., 2012). Penelitian lain pada (Wahyu W et al., 2014) telah menganalisa bagaimana performa *avaibility* prediksi *auto scaling cloud computing* dengan menggunakan *moving average*. Pada (Calheiros et al., 2014) prediksi *workload* untuk *auto scaling cloud computing* dengan penambahan *autoregressive integrated* pada *moving average*, hasilnya menunjukkan rata-rata akurasi adalah 91%. Dari berbagai literatur belum ditemukan metode kombinasi antara *moving average* dan *real time* digunakan untuk optimasi daya menggunakan DNS. Namun dari berbagai penelitian tersebut memiliki kesamaan dengan penelitian ini, maka jelas sekali bahwa metode *moving average* yang dikombinasikan dengan *real time* memungkinkan dapat diterapkan pada optimasi daya dengan skema DNS.

Hipotesis yang akan didapatkan dengan menggunakan metode prediksi *moving average* dari penelitian ini adalah dapat melakukan optimasi daya pada *cloud computing* tanpa mempengaruhi performa dari layanan tersebut.

1.2 Perumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan. Adapun perumusan masalah dari penelitian ini adalah:

1. Bagaimana mengoptimasi konsumsi daya dengan migrasi VM (*Virtual Machine*) dan DNS (*Dynamics Shutdown*)
2. Bagaimana membuat penjadwalan migrasi dan DNS secara otomatis
3. Bagaimana menerapkan metode prediksi m*oving average* secara *real time* pada penjadwalan migrasi VM dan DNS
4. Bagaimana melakukan evaluasi menggunakan parameter konsumsi daya dan *response time*.

1.3 Kontribusi Penelitian

Kontribusi pada penelitian ini adalah membuat mekanisme penjadwalan atau *scheduling* menggunakan pendekatan metode *moving average* yang akan dikombinasikan dengan *real time* agar dapat diterapkan dan digunakan untuk optimasi penghematan energi pada data *center cloud computing* dengan migrasi otomatis dan DNS serta dapat mengurangi pengaruh pada performanya.

1.4 Tujuan dan Manfaat

Tujuan dan manfaat penelitian ini adalah melakukan optimasi daya pada data *center cloud computing* dengan cara migrasi otomatis dan DNS menggunakan kompinasi metode prediksi dan *real time* sehingga konsumsi daya pada setiap server dan keseluruhan server ataupun data *center cloud computing* dapat dikurangi sampai titik terendah tanpa mempengaruhi performa data *center*.

1.5 Batasan Penelitian

Berdasarkan beberapa uraian yang telah dijelaskan sebelumnya, batasan masalah pada penelitian ini adalah:

1. Optimasi daya pada data *center* dilakukan dengan migrasi VM (*Virtual Machine*) dan DNS
2. Lingkungan uji coba penelitian ini memanfaatkan *tool virtual machine* VMware Workstation 12.1.1 build-3770994

3. *Cloud computing* yang dibangun dalam *virtual machine* VMware Workstation menggunakan sistem operasi Proxmox versi 4.1-1
4. VM (*Virtual Machine*) yang berada dalam *cloud computing* menggunakan sistem operasi Ubuntu Server 14.04.4
5. Metode penjadwalan yang digunakan adalah menggunakan kombinasi metode atau algoritma prediksi dan *real time*
6. Data prediksi yang diambil adalah dari *usage* CPU server VM yang berada pada server *cloud computing.*

BAB 2
KAJIAN PUSTAKA DAN DASAR TEORI

2.1 Kajian Pustaka

Perkembangan *cloud computing* memiliki garis lurus dengan perkembangan data *center*, semakin berkembang data *center* maka semakin banyak tambahan server yang dibutuhkan, sehingga semakin banyak pula konsumsi energi yang dibutuhkan untuk mengoptimalkan layanan *cloud computing* tersebut.

Optimasi daya pada data *center cloud computing* dengan kombinasi prediksi dan *real time* merupakan solusi tepat untuk menangani masalah yang ditimbulkan oleh perkembangan pesat *cloud computing*. Mencegah kegagalan optimasi energi pada data *center* dengan DNS dapat dilakukan dengan penjadwalan model prediksi menggunakan metode *moving average* dan *real time*.

Dalam perkembangannya sistem optimasi daya telah banyak dilakukan pada berbagai penelitian. Di antaranya adalah seperti yang dilakukan oleh (Choi et al., 2004) optimasi yang dilakukan adalah dengan cara mengurangi konsumsi daya pada CPU dilakukan dengan skema *Dynamic Voltage and Frequency Scaling* (DVFS) telah menunjukkan penghematan energi CPU hingga mencapai 80%. Pada penelitian (Kliazovich et al., 2012) telah mengkaji perbandingan optimasi daya dengan DVFS, DVFS + DNS (*Dynamics Shutdown*) hasilnya menunjukkan penghematan lebih baik.

Penelitian mengenai *scheduling* atau penjadwalan pada optimasi daya telah dilakukan pada beberapa paper seperti (Lee, 2009) menggunakan metode penjadwalan *real time*, (Lakshmanan et al., 2010) menggunakan *scheduling parallel real time*.

Penelitian mengenai metode prediksi atau prediktif pada *cloud computing* biasanya sering dilakukan pada penelitian *auto scaling*, seperti pada (Yang et al., 2013), (Roy et al., 2011) membuat penelitian tentang prediksi *auto scaling* pada virtual agar dapat memprediksi *workload cloud computing*, namun belum ada penelitian untuk penjadwalan optimasi daya menggunakan model prediksi atau kombinasi antara prediksi dan *real time*.

Penelitian mengenai *moving average* sebagai metode atau algoritma prediksi, banyak digunakan untuk memprediksi fluktuasi saham seperti pada penelitian (Zulkarnain, 2012). Sebagaimana kita ketahui, saham memiliki fluktuasi yang sangat dinamis, namun dengan metode prediksi seperti penelitian (Zulkarnain, 2012) menunjukkan hasil yang sangat baik dalam mendeteksi perkembangan saham. Dalam penelitian *cloud computing* metode *moving average* digunakan sebagai metode prediksi pada *auto scaling* (Lorido-Botran et al., 2012). Penelitian lain pada (Wahyu W et al., 2014) telah menganalisa bagaimana performa *avaibility* predidiksi *auto scaling cloud computing* dengan menggunakan *moving average.* Parameter-parameter yang digunakan pada beberapa penelitian tersebut memiliki kesamaan dengan penelitian ini, maka jelas sekali bahwa Metode *moving average* yang dikombinasikan dengan *real time* memungkinkan dapat diterapkan pada optimasi daya pada penelitian ini.

2.2 Dasar Teori

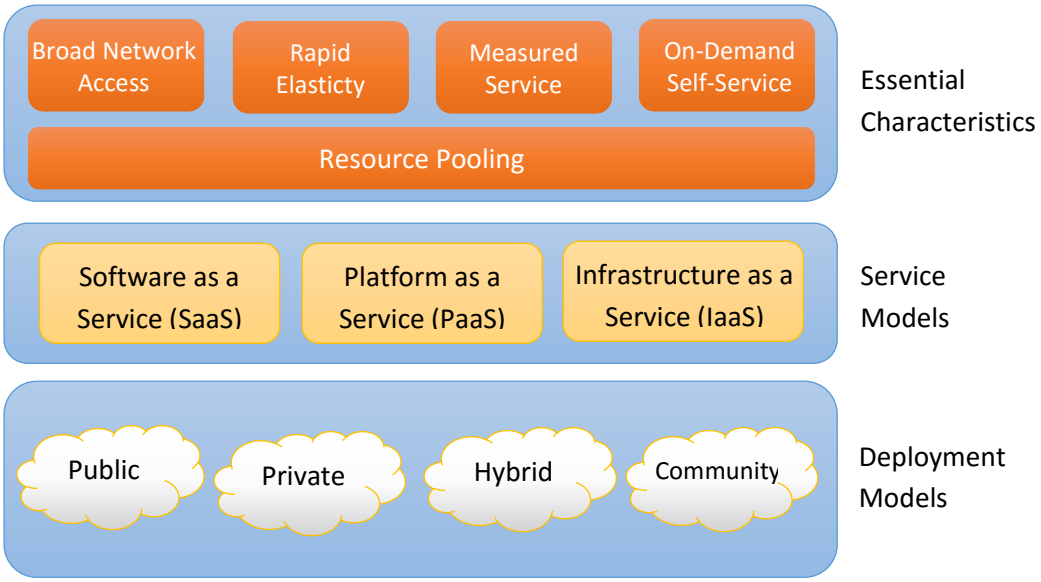
Sub-bab dasar teori akan memaparkan teori-teori dasar yang digunakan untuk mendukung penelitian yang dilakukan. Dasar teori yang diuraikan meliputi pengertian *cloud computing*, data *center*, prosesor, optimasi daya pada data *center*, algoritma *moving average.*

2.2.1 *Cloud computing*

Komputasi awan atau *cloud computing* adalah gabungan antara pemanfaatan teknologi komputer dengan pengembangan berbasis *internet*. Awan (*cloud*) adalah metafora dari *internet*, sebagaimana awan yang sering digambarkan pada diagram jaringan komputer (Jati Waloeyo, 2012).

Menurut penjelasan dalam buku (Sofana, 2012) *cloud computing* adalah sebuah model *client*-server, di mana *resource* seperti server, *storage, network,* dan *software* dapat dipandang sebagai layanan yang dapat diakses oleh pengguna secara *remote* dan setiap saat. Pengguna dapat menikmati berbagai layanan yang disediakan oleh *provider cloud computing*, tanpa perlu terlalu banyak meminta bantuan teknis atau *support* dari pihak *provider*. Infrastruktur *cloud computing* seperti server, *storage, network* dan berbagai *software* disebut "*cloud".*

Sedangkan mnurut *National Standards and Technology* (NIST) (Mell and Grance, 2011), "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models*".



Gambar 2.1 Bagan c*loud computing* menurut NIST (Mell and Grance, 2011)

Cloud Computing merupakan suatu metode komputasi di mana kapabilitas terkait teknologi informasi disajikan sebagai suatu layanan, sehingga pengguna dapat mengaksesnya lewat *internet* tanpa mengetahui apa yang ada di dalamnya, ahli dengannya, atau memiliki kendali terhadap infrastruktur teknologi yang membantunya. Menurut NIST dalam buku karangan Iwan Sofana (Sofana, 2012) ada lima karakteristik *cloud computing* yaitu:

1. *Rapid Elasticity*

   Resource yang disediakan oleh *cloud computing* dapat bertambah atau berkurang (*scale up and down*) dengan cepat.

2. *Measured Service*

   Services yang disediakan bersifat terukur. *Provider cloud computing* dapat mengendalikan dan memonitor *cloud services*, misalkan untuk keperluan *billing, access control, resource optimization, capacity planning,* dan sebagainnya.

3. *On-Demand Self-Service*

   Pengguna dapat mengakses *cloud computing services* sesuai kebutuhan, tanpa perlu dilayani oleh pihak lain (misal: teknisi *provider cloud computing*).

4. *Ubiquitous/Broad Network Access*

   Semua kapasitas *cloud computing* tersedia melalui *network* dan dapat diakses oleh *clients* (*mobile device, thick* atau *thin client*) dengan metode yang sudah berlaku secara umum.

5. *Resource Pooling*

   Cloud computing provider dapat melayani pengguna via *multi-tenant model*. Berbagai *resource*, seperti: *storage,* CPU, *memory*, *bandwidth*, dan mesin virtual (*virtual machine*), yang terdapat di berbagai lokasi dapat digunakan oleh banyak *client* secara bersamaan.

   Kemudian, ada tiga *service layer* atau *delivery model* yang disediakan *cloud computing*:

   - *Infrastructure as a Service* (IaaS)

     Pengguna dapat menggunakan *fundamental computing resource* seperti *processing power, storage, networking component*. Pengguna diizinkan untuk menginstal sistem operasi, *storage,* membangun aplikasi sendiri, membuat *firewall* dan *load balancer*. Contoh IaaS adalah Amazon *Elastic Compute Cloud.*

   - *Platform as a Service* (PaaS)

     Pengguna dapat mengembangkan aplikasi menggunakan *application framework* atau *application engine* yang disediakan oleh *provider*. Pengguna dapat secara leluasa mengontrol aplikasi, namun tidak dapat mengontrol sistem operasi, *hardware,* atau *network*. Contoh PaaS yaitu Force.com dan Microsoft Azure Investment.

- *Software as a Service*

  Pengguna dapat menggunakan aplikasi namun tidak dapat membuat aplikasi, tidak dapat mengontrol sistem operasi, *hardware*, dan *network*. Aplikasi dapat diakses via *Web-Browser* atau *Web based interface*. Contoh SaaS adalah GoogleDoc dan SalesForce.

Dari sifat-sifat *cloud computing* yang dijelaskan di atas maka sangat jelas bahwa *cloud computing* merupakan sistem yang harus memiliki performa tinggi atau sering disebut sebagai *High Performance Computing* (HPC). Sebenarnya HPC akan bertolak belakang dengan penghematan konsumsi energi, karena semakin tinggi performa *computing* maka semakin tinggi pula konsumsi energi yang dibutuhkan oleh sistem. Oleh karena itu, penelitian ini bertujuan untuk menyelesaikan masalah tersebut dengan cara mengoptimasi penghematan konsumsi daya namun tetap menjaga performa dari *cloud computing* tersebut.

2.2.2 Data *Center*

Menurut Sofana dalam bukunya (Sofana, 2012), data *center* adalah suatu fasilitas yang digunakan untuk menempatkan sistem komputer dan komponen-komponen terkaitnya, seperti sistem telekomunikasi dan penyimpanan data. Fasilitas ini biasanya mencakup juga catu daya redundan (cadangan), koneksi komunikasi data redudan, pengontrol lingkungan (AC, ventilasi), pencegah bahaya kebakaran, serta piranti keamanan fisik.

Data *center* merupakan kunci utama yang menjadi tolak ukur penilaian sebuah *provider cloud computing*, semakin baik data *center* maka semakin baik pula layanan *cloud computing* yang dihasilkan. Data *center* memiliki tingkatan atau *level* tersendiri yang disebut *tier*, menurut Sofana (Sofana, 2012) ada sebuah lembaga yang telah membuat spesifikasi *tier* untuk data *center*, yaitu *Telecommunications Industry Association* (TIA). TIA telah terakreditasi oleh ANSI (*American National Standards Institute*). Pada tahun 2005, TIA mempublikasikan ANSI/TIA-942, yaitu *Telecommunications Infrastructure Standard for* Data *Centers*, yang mendifinisikan empat tingkatan (*tier*) data *center*. Berikut ini adalah table yang berisi spesifikasi masing-masing *tier* data *center*.

Tabel 2.1 Tingkatan (*level*) *Tier* pada Data *Center* Menurut TIA

Tier	Requirement
1	➢ Komponen semuanya nonredundan (server, *link,* dan lain-lain ➢ Garansi ketersediaan (*availability*) sebesar 99,671%
2	➢ Mencakup semua aspek *Tier*-1 ➢ Komponen sudah redundan ➢ Garansi ketersediaan (*availability*) sebesar 99,741%
3	➢ Mencakup semua aspek *Tier*-1 dan *Tier*-2 ➢ Sumber listrik ada dua (*dual powered*) ➢ Link banyak (*multiple*) ➢ Semua perangkat harus memiliki *dual power supply* dan kompatibel dengan arsitektur lokasi yang mencakup rak dan pengkabelan ➢ Garansi ketersediaan (*availability*) sebesar 99,982%
4	➢ Mencakup semua aspek *Tier*-1 dan *Tier*-2 dan *Tier*-3 ➢ Semua perangkat pendinginan memiliki *dual power supply*, termasuk *chillers*, *heating*, *ventilating* dan *air-conditioning* (HVAC) *systems* ➢ Garansi ketersediaan (*availability*) sebesar 99,995%

Terlihat di dalam Tabel 2.1 mengenai ketersediaan (*availability*) dalam persen, seperti *Tier*-4 memiliki garansi ketersediaan sebesar 99,995% ini berarti data *center* dan semua perangkat yang ada harus dapat beroprasi sebanyak 99,995% per tahun. Artinya data *center* hanya boleh mengalami *down time* sebesar (100%-99,995%)=0,005%.

Jika angka 0,005% dikonversikan dalam bentuk menit, maka kurang lebih *down time* yang boleh terjadi atau diizinkan untuk data *center* hanya 43,8 menit per tahun, ini merupakan jumlah yang sangat kecil sekali jika dibandingkan dengan jumlah menit dalam satu tahun yaitu 525.600 Menit = 8.760 Jam.

Kesalahan dalam optimasi daya pada data *center* dapat berpengaruh terhadap performa data *center*, sekaligus akan menyebabkan *down time* data *center* semakin meningkat. Jika melihat persyaratan sebuah data *center* maka jelas sekali performa server atau data *center* harus tetap dijaga walaupun konsumsi daya pada data *center* tersebut telah dikurangi. Sehingga perlu adanya mekanisme optimasi konsumsi daya yang seiring dengan kebutuhan persyaratan data *center* tersebut.

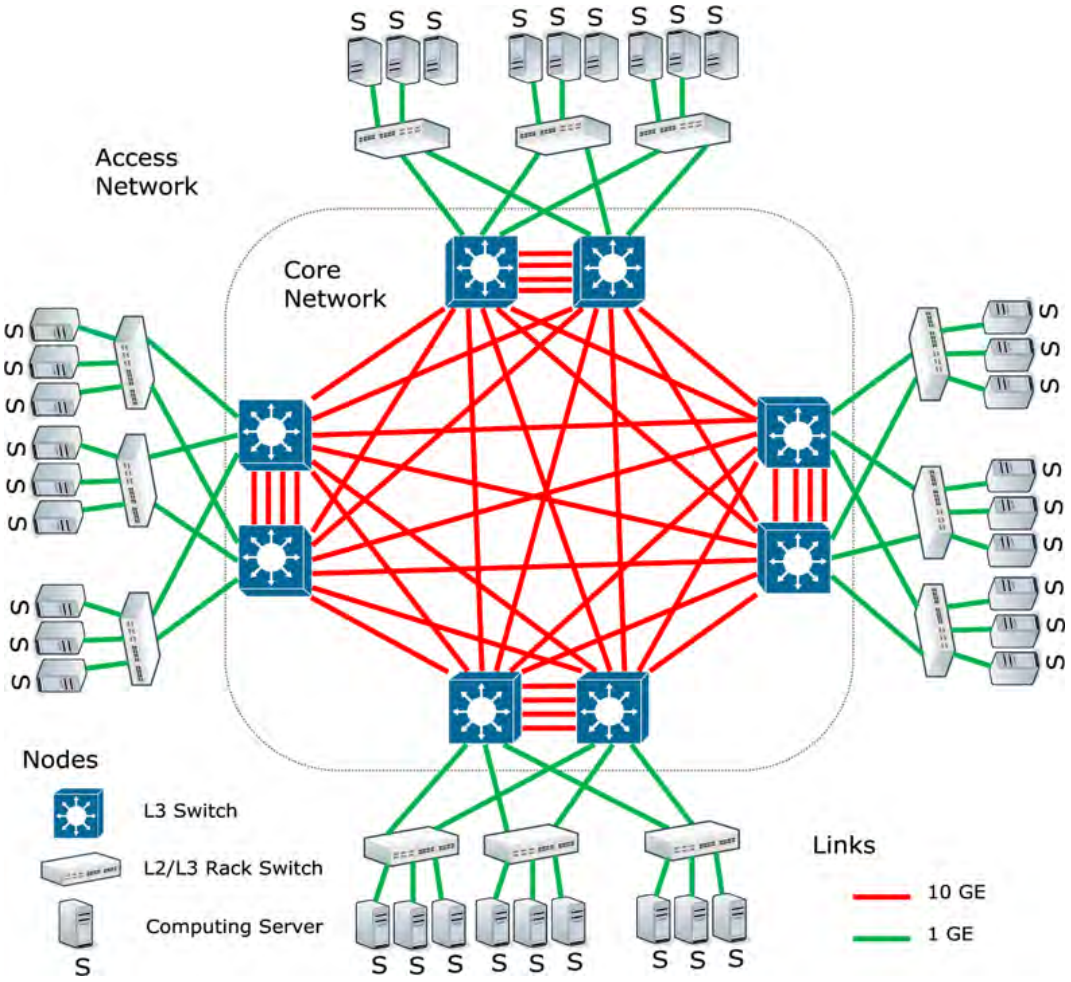
2.2.2.1 Arsitektur Data *Center*

Sebuah data *center* saat ini mampu menangani sampai dengan 100.000 *host* dengan sekitar 70% pelaksanaan komunikasi dilaksanakan secara internal (Mahadevan et al., 2009). Hal ini memberikan tantangan dalam merancang arsitektur jaringan interkoneksi dan protocol komunikasinya.

Telecommunications Industry Association (TIA) mempublikasikan ANSI/TIA-942, yaitu *Telecommunications Infrastructure Standard for* Data *Centers*, yang mendifinisikan empat tingkatan (*tier*) data *center* ("TIA-942 Data Centre Standards Overview", 2008). Dalam penelitian (Kliazovich et al., 2012) telah menjelaskan beberapa arsitektur data *center.* Di antaranya yaitu arsitektur data *center two-tier*, *three-tier*, dan *three-tier high-speed* data *center*.

Berdasarkan struktur yang diperlihatkan pada Gambar 2.2, pada arsitektur *two-tier*, *Computing* Server (S) disusun ke dalam rak membentuk jaringan *tier-one*. Pada jaringan *tier-two, switch* pada *Layer*-3 (L3) menyediakan konektivitas mesh penuh menggunakan *link* 10 GE.

Arsitektur *two-tier* telah bekerja dengan baik pada data *center* generasi awal di mana jumlah server yang masih terbatas. Tergantung dari jenis *switch* yang dipakai pada jaringan akses (*access network*), arsitektur *two-tier* dapat mendukung sampai dengan 5500 *node*. Jumlah dari *core switch* dan kapasitas dari *core link* menyatakan *bandwidth* jaringan maksimum yang dialokasikan pada setiap server.

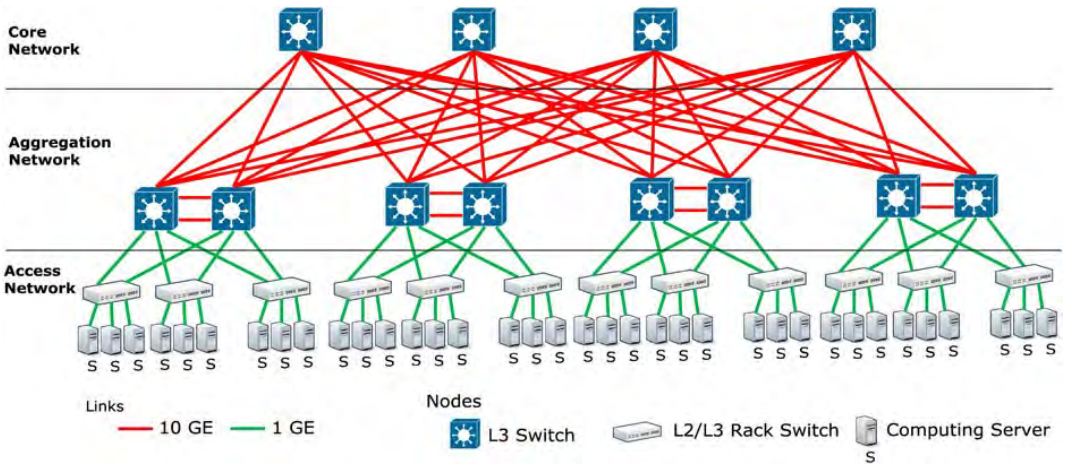


Gambar 2.2 Arsitektur Data *Center Two-tier* (Kliazovich et al., 2012)

Arsitektur *three-tier* adalah yang paling banyak dipakai saat ini. Arsitektur ini terdiri atas lapisan: *access*, *aggregation* dan *core*. Keberadaan lapisan *aggregation* meningkatkan jumlah *node* server (lebih dari 10000 server) dengan tetap menjaga *Layer*-2 menggunakan *switch* yang tidak terlalu mahal pada jaringan *access* yang menyediakan topologi *loop-free*.

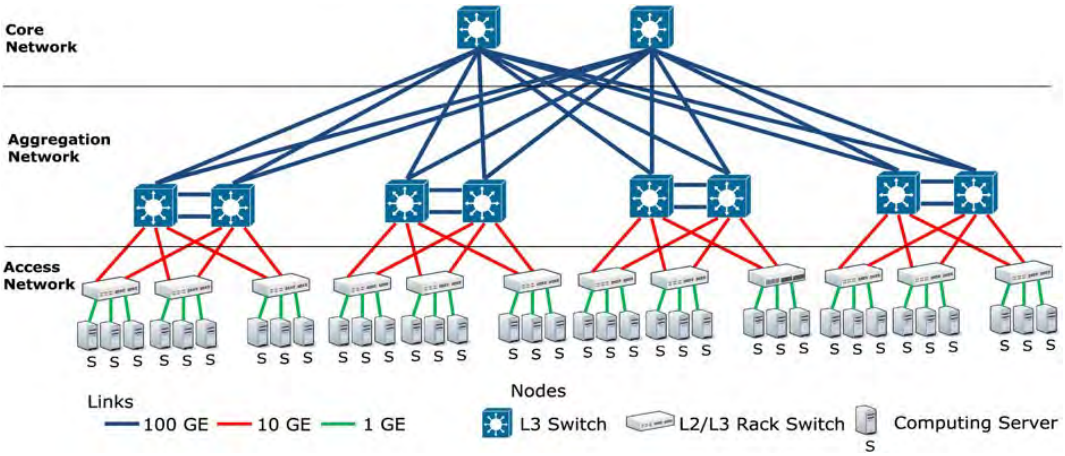
Karena jumlah maksimum jalur pada ECMP yang diijinkan adalah delapan, maka pada arsitektur *three-tier* terdiri atas delapan *core* (diperlihatkan hanya empat pada Gambar 2.3). Arsitektur ini menerapkan 8 jalur ECMP yang sudah termasuk ke dalamnya 10 GE *Line Aggregation Groups* (LAGs) yang memungkinkan sebuah jaringan client mengalamati beberapa *link* dan *port* jaringan dengan alamat MAC tunggal.

Sementara teknologi LAG adalah sebuah metodologi yang sangat baik bagi peningkatkan kapasitas dari *link*, namun kegunaannya memiliki beberapa kekurangan yang akan membatasi fleksibilitas jaringan dan unjuk kerja. LAG mengakibatkan sangat sulit untuk membuat perencanaan kapasitas untuk aliran data yang besar dan membuatnya tak dapat diperkirakan dalam hal hal kegagalan link. Lebih lanjut lagi, beberapa macam pola trafik misalnya ICMP dan *broadcast* biasanya dirutekan hanya melalui jalur tunggal. Belum lagi untuk konektivitas *full mesh* pada lapisan *core* pada jaringan memerlukan sejumlah pertimbangan dalam melakukan pengkabelan.



Gambar 2.3 Arsitektur Data *Center Three-tier* (Kliazovich et al., 2012)

Arsitektur Data *Center Three-tier High-speed* dirancang untuk mengoptimalkan jumlah *node*, kapasitas *core* dan jaringan aggregasi yang biasa mengalami *bottleneck*. Arsitektur ini hampir mirip dengan *three-tier* hanya saja link antara *core* dan aggregation memiliki kapasitas 100 GE seperti tampak pada Gambar 2.4 berikut.



Gambar 2.4 Arsitektur Data *Center Three-tier High-speed* (Kliazovich et al., 2012)

Dengan kapasitas link antara *core* dan *aggregation* sebesar 100 GE maka akan mengurangi jumlah *switch* pada *core*, menghindari kekurangan teknologi LAG, mengurangi pengkabelan, dan meningkatkan ukuran maksimum dari data *center* akibat adanya keterbatasan fisik. Lebih sedikit jalur ECMP akan lebih fleksibel dan meningkatkan unjuk kerja jaringan.

Sebenarnya data *center* merupakan arsitektur kumpulan komputer yang sangat kompleks, seperti terlihat dari beberapa arsitektur di atas menunjukkan bahwa segala macam jenis *hardware* akan diperhitungkan, seperti pengkabelan, switch, dan lain-lain. Namun pada arsitektur yang akan dibangun pada penelitian ini tidak akan dapat memenuhi kompleksitas arsitektur data *center* sebenarnya seperti penjelasan di atas. Penelitian ini akan berfokus hanya pada server saja, namun arsitekturnya akan didesain semirip mungkin dengan data *center cloud* yang sbenarnya dengan mempertimbangkan *hardware* yang tersedia untuk pengujian.

2.2.3 Prosesor

Komponen prosesor mengkonsumsi sebagian besar energi dari sebuah sistem, penghematan energi pada komponen prosesor adalah lebih penting daripada biaya *hardware* pada banyak kasus (Lee, 2009).

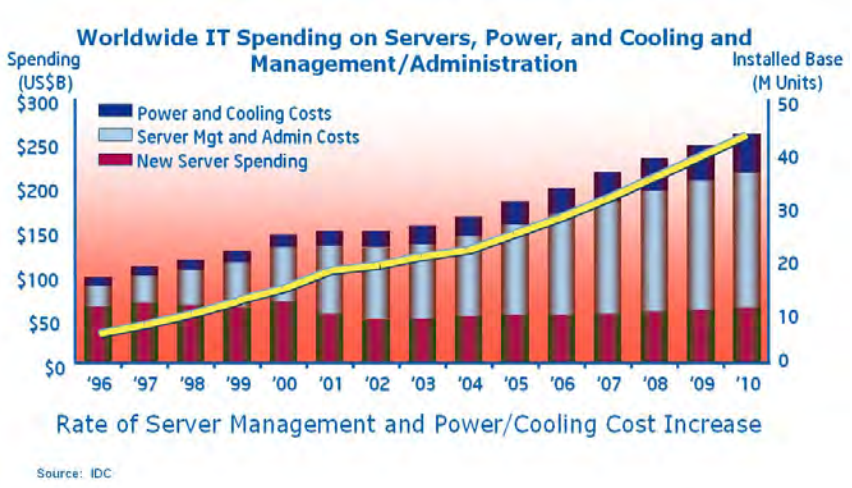
Prosesor sebagai bagian dari perangkat IT pada data *center*, akhir-akhir ini sudah mulai dirancang bukan hanya menghasilkan unjuk kerja yang baik namun

memiliki kemampuan untuk menghemat konsumsi energinya. Hal ini menjadi tujuan karena untuk mendapatkan unjuk kerja prosesor yang tinggi tidak hanya memerlukan konsumsi energi yang banyak akan tetapi juga menghasilkan panas yang besar. Hal ini mendorong para insinyur untuk mengekploitasi dari aspek fisik untuk meningkatkan efisiensi. Saat ini CPU komputer sudah memiliki mekanisme penghematan energi dengan menurunkan laju *clock* melalui skema yang disebut *Dynamic Voltage Scaling* atau DVS dan juga kemampuan mematikan sebagian dari chip melalui mekanisme yang disebut skema *Dynamic Power Management* atau DPM.

Choi dalam risetnya (Choi et al., 2004) menjelaskan bahwa mikroprosesor modern sudah dilengkapi fungsi optimasi daya menggunakan skema DVFS, seperti Intel's XScale (Intel, 2003).

2.2.4 Optimasi Daya pada Data *Center*

Masalah utama dari pesatnya perkembangan komputasi, baik itu komputasi individual maupun komputasi sekelas data *center provider cloud computing* adalah meningkatnya konsumsi energi. Seperti terlihat dalam Gambar 2.5 di bawah ini terlihat bahwa daya dan pendinginan pada data *center* cenderung memiliki biaya yang terus meningkat seiring dengan perkembangan waktu.



Gambar 2.5 Struktur Pembiayaan Data *Center* (Filani et al., 2008)

Untuk mengatasi hal ini, pengelola data *center* memiliki beberpa pilihan di antaranya sebagai berikut (Filani et al., 2008):

1. Menambah kapasitas daya dan pendingin
2. Membangun data *center* baru
3. Melakukan pengelolaan energi yang memaksimalkan kapasitas yang ada.

Dua pilihan awal akan sangat mahal karena melibatkan belanja modal dan instalasi baru. Maka pilihan ketigalah yang paling memungkinkan untuk mengatasi dua hal tersebut di atas.

2.2.4.1 *Dynamics Shutdown* (DNS)

Dalam sebuah penelitian menjelaskan bahwa server yang dalam kondisi *idle* akan tetap mengkonsumsi energi sebesar 66% dari kapasitas penuhnya (Kliazovich et al., 2012), maka pada mekanisme DNS, skema penghematan dilakukan adalah dengan cara mematikan server yang dalam kondisi *idle* sehingga konsumsi energi bisa ditekan pada kondisi minimal.

Pemanfaatan DNS oleh (Ikram et al., 2013) telah membahas penghematan energi pada *Storage Area Networks* (SAN) mencapai 35,6 %. Sedangkan dalam penelitian (Kliazovich et al., 2012) telah membahas perbandingan DVFS, DNS, dan DVFS+DNS seperti yang ditunjukkan pada Tabel 2.2.

Tabel 2.2 Perbandingan Skema Efisiensi Energi (Kliazovich et al., 2012)

Paramter	**Power consumtion (kWh)**			
	No energy-saving	**DVFS**	**DNS**	**DVFS+DNS**
Data center	503.4	486.1 (96%)	186.7 (37%)	179.4 (35%)
Servers	351	304.5 (97%)	138.4 (39%)	132.4 (37%)
Switches	152.4	145.6 (95%)	48.3 (32%)	47 (31%)
Energy cost/year	$441k	$435k	$163.5k	$157k

Pada Tabel 2.2 menunjukkan bahwa penghematan energi dengan menggunakan skema DNS hanya menghabiskan biaya $163.5k dari pada yang tidak menggunakan skema sebesar $441 dan dengan skema DVFS adalah $435k. dan akan lebih bagus lagi jika DNS dikombinasi dengan DVFS yaitu hanya $157k. Ini

artinya skema DNS menunjukkan hasil yang sangat bagus dalam hal optimasi konsumsi daya.

2.2.5 Metode *Moving Average*

Moving average merupakan salah satu jenis metode prediksi berdasarkan *time series* atau keturutan waktu kuantitatif dalam teori peramalan. Metode *moving average* menggunakan nilai di masa lalu yang digunakan sebagai acuan dalam melakukan prediksi di masa depan. Secara umum tujuan dari jenis peramalan *time series* adalah menemukan pola dalam deret historis dari suatu data dan mengeskploitasinya untuk dijadikan pola masa depan.

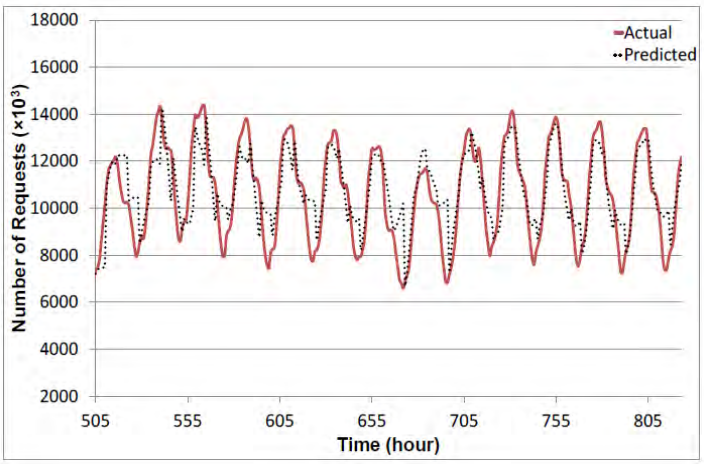
Data *time series* seringkali mengandung ketidak teraturan yang akan menyebabkan prediksi yang beragam. Untuk menghilangkan efek yang tidak diinginkan dari ketidak teraturan ini, metode *moving average* mengambil beberapa nilai yang sedang diamati, memberikan rataan, dan menggunakannya untuk memprediksi nilai untuk periode waktu yang akan datang (Yaffee and McGee, 2000). Semakin banyak jumlah pengamatan yang dilakukan, maka pengaruh metode *moving average* akan lebih baik. Meningkatkan jumlah observasi akan menghasilkan nilai peramalan yang lebih baik karena ia cenderung meminimalkan efek-efek pergerakan tidak biasa yang muncul pada data. Persamaan matematis *single moving average* adalah sebagai berikut (Wahyu W et al., 2014).

$$A_t = \frac{D_t + D_{t-1} + D_{t-2} + \cdots + D_{t-N+1}}{N} \qquad (2.1)$$

Di mana:
N adalah total jumlah periode rataan.
A_t adalah prediksi pada periode t + 1
D_t adalah adalah nilai *real* periode ke t

Persamaan di atas merupakan persamaan sederhana *moving average,* namun jika persamaan tersebut diberikan penambahan *autoregressive, maka* dapat meningkatkan akurasi prediksi dengan *time series* (Calheiros et al., 2014). Model *moving average* dengan penambahan *autoregressive* sering disebut dengan model *Box-Jenkins* (ARIMA) dibagi ke dalam 3 kelompok, yaitu: model autoregressive

(AR), *moving average* (MA), dan model campuran ARIMA (*autoregresive moving average*) yang mempunyai karakteristik dari dua model pertama. ARIMA adalah metode untuk prediksi dengan data *non-stasioner* yang terdiri dari *autoregresif* dan model *moving average*, dan berhasil dimanfaatkan untuk prediksi *time series* di domain yang berbeda seperti keuangan (Calheiros et al., 2014). Berikut adalah hasil yang diperlihatkan dalam penelitian (Calheiros et al., 2014) mengenai *autoregresive moving average*.

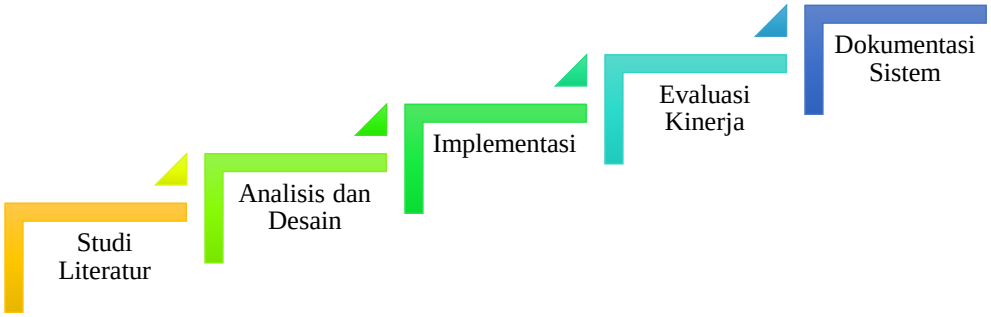


Gambar 2.6 Hasil Prediksi *Autoregresif Moving Average* (Calheiros et al., 2014)

Dari Gambar 2.6 mengenai grafik perbandingan data prediksi dengan aktual di atas menunjukkan bahwa data prediksi memiliki keakuratan tinggi terhadap data aktual, hanya memiliki *margin error* atau selisih yang sangat sedikit.

BAB 3
METODE PENELITIAN

Bab ini memaparkan tentang metodologi penelitian yang akan digunakan pada penelitian ini, antara lain terdiri dari (1) studi literatur, (2) desain dan implementasi, (3) analisis pengujian, (4) evaluasi kinerja, (5) dokumentasi sistem dan jadwal kegiatan, untuk ilustrasi alur metodelogi penelitian bisa dilihat pada Gambar 3.1.



Gambar 3.1 Alur Metode Penelitian

3.1 Studi Literatur

Studi literatur yang dilakukan bertujuan untuk memperoleh informasi yang berkaitan dengan penelitian yang akan dilakukan, yaitu:

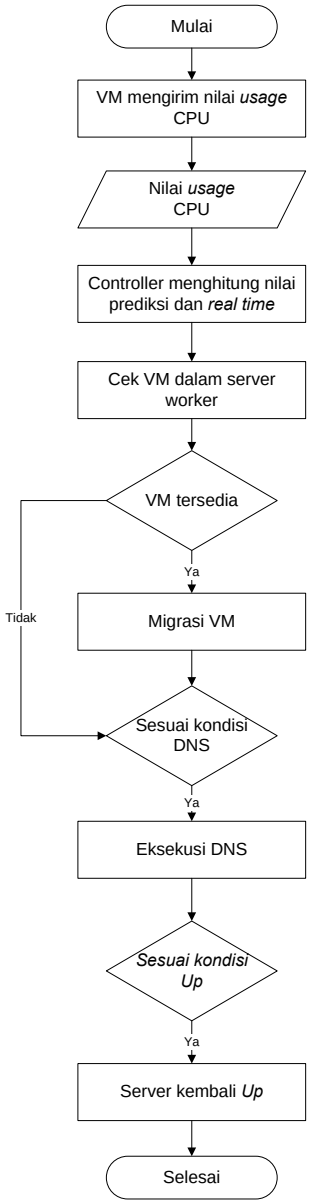
a. Mengetahui mekanisme-mekanisme yang pernah digunakan pada optimasi daya data *center cloud computing* yang menggunakan DNS.
b. Mengetahui metode prediksi yang dapat digunakan pada DNS.
c. Mengetahui algoritma prediksi *moving average* sehingga dapat mengoptimasi daya secara *real time.*
d. Mengetahui bagaimana pengaruh optimasi daya terhadap performa data *center*.

3.2 Analisis dan Desain

Pada penelitian ini diusulkan desain sistem optimasi daya data *center cloud computing* menggunakan *Dynamics Shutdown* (DNS) dengan metode prediksi *time series moving average* secara *real time.* Penelitian ini diharapkan mampu mengurangi konsumsi daya server serta dapat menjaga performa server.

3.2.1 *Flowchart* Sistem yang Diusulkan

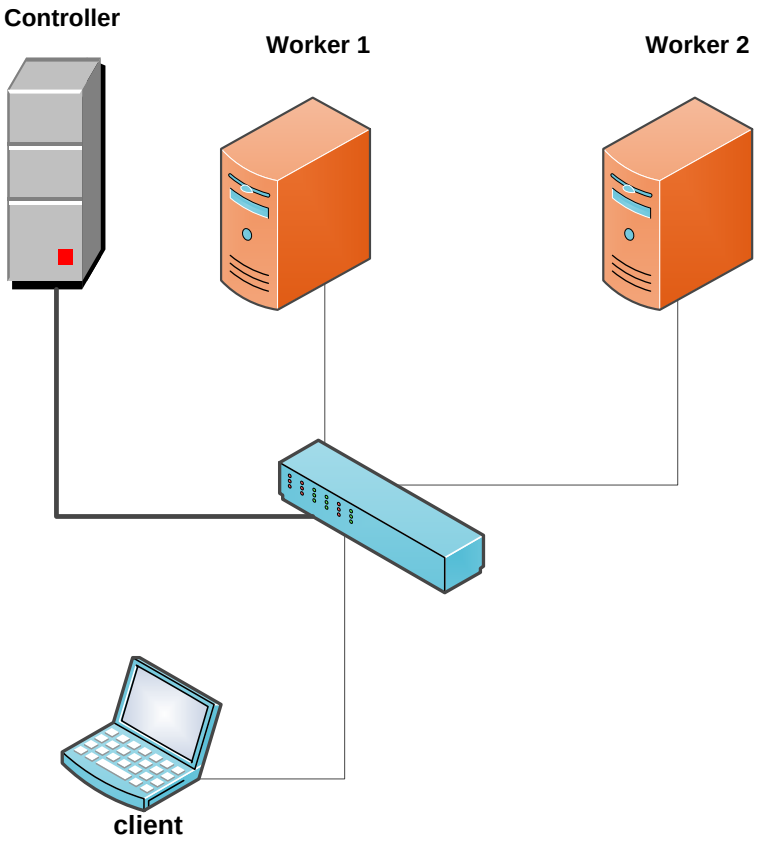
Sub-bab ini memaparkan *flowchart* dari sistem yang diusulkan, *flowchart* yang dipaparkan pada bab ini merupakan *flowchart* sistem secara keseluruhan untuk mengoptimasi daya dengan DNS sesuai penjelasan kondisi yang telah dijelaskan sebelumnya.



Gambar 3.2 *Flowchart* Sistem yang Diusulkan

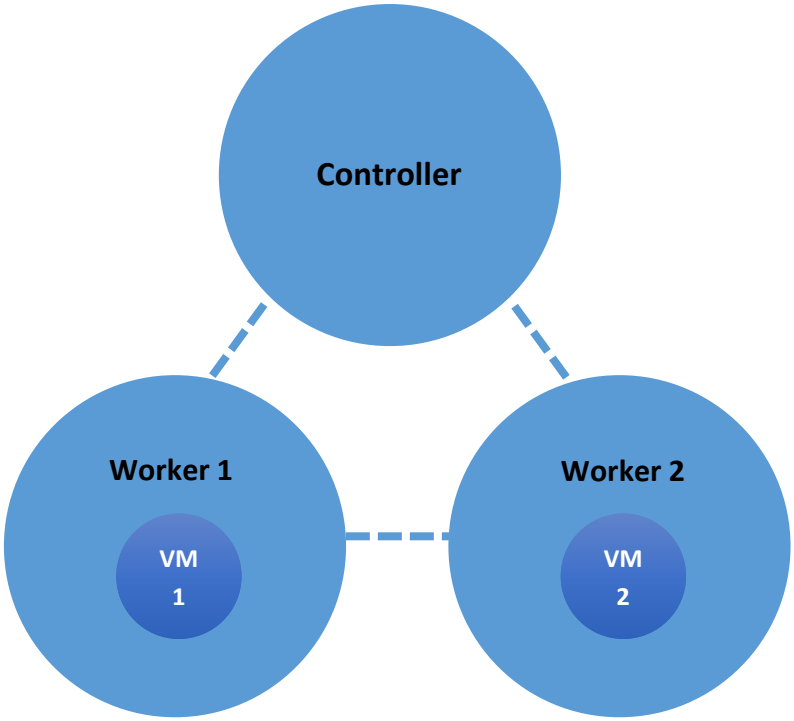
3.2.2 Desain Arsitektur *Cloud Computing* yang Diusulkan

Arsitektur *cloud computing yang diusulkan* terdiri dari cluster server, dimana terdapat satu server *controller* dan dua server *worker*.



Gambar 3.3 Arsitektur C*loud Computing* sebagai Pengujian Sistem

Dalam hal penelitian ini akan digunakan beberapa server seperti yang terlihat pada rancangan Gambar 3.3 di atas. Server terdiri dua server *worker*. Pengujian sistem yang diusulkan pada penelitian ini hanya berfokus pada lingkungan server, artinya penelitian ini tidak akan memasukkan *core network*, *link state*, dll sebagai mana data *center cloud computing*. Namun sistem yang diusulkan ini akan didesain semirip mungkin dengan data *center cloud computing*, sistem *cloud computing* pada pengujian ini akan memanfaatkan *framework* yang telah tersedia sebagai virtualisasi.



Gambar 3.4 Sistem *Cloud Computing* yang Dibangun

Hubungan antara *controller, worker* dan VM dapat dilihat seperti pada Gambar 3.4. Masing-masing *worker* memiliki VM di dalamnya, setiap VM dapat berpindah dari *worker* 1 ke *worker* 2, dan begitu juga sebaliknya. *Controller, worker*, dan VM saling terhubung, sehingga *controller* dapat mengatur semua entitas yang ada pada sistem.

3.2.3 Pengukuran *usage* CPU

CPU (Central Processing Unit) *usage* atau CPU *Time* adalah metode untuk mengukur kuantitas penggunaan CPU, khususnya metode untuk mengukur kuantitas penggunaan CPU yang mampu mendapatkan jumlah kredibel penggunaan CPU tanpa mengubah algoritma dalam rangka untuk beradaptasi dengan sistem operasi, misalnya, MS-Windows System, atau memerlukan kode yang rumit. Metode ini menggunakan berbagai algoritma yang disediakan oleh sistem operasi pada nama registri menyimpan kuantitas penggunaan CPU di dalam sistem. Dengan

demikian penemuan ini dapat mengukur kuantitas penggunaan CPU dengan mudah tanpa menurunkan kinerja dari sistem operasi (Ho Lee and Keun Oh, 2004).

Idle Ticks = Sum (across sampling interval) [Second Timer Read − Initial Timer Read]

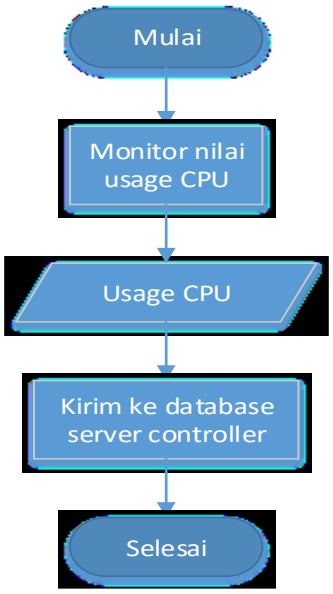
$$\text{CPU Idle}(\%) = \text{Idle Ticks} \times \frac{Tick\ Period(s)}{Sampling\ Interval} \times 100\% \quad (3.1)$$

Pada persamaan 3.1 dapat dilihat CPU *idle* dalam bentuk satuan persen, maka yang akan dihitung adalah penggunaan CPU yang merupakan hasil pengurangan dari 100% dikurangi dengan CPU *idle* dalam bentuk satuan persen diukur dari sisi *server*. Evaluasi akan membandingkan bagaimana hubungan konsumsi daya pada sistem yang menggunakan metode prediksi dan bagaimana hubungan konsumsi daya tanpa menggunakan metode prediksi.

Usage CPU merupakan hal penting yang akan diperhitungkan dalam penelitian ini. Pada penelitian ini *usage* CPU akan digunakan sebagai data *real time* dan prediksi menggunakan *time series moving average*. Data *usage* CPU yang digunakan adalah data *usage* yang terdapat pada server VM.

Usage CPU dari VM akan dikirim secara *real time* ke server *controller*, setelah data *usage* CPU diterima, maka server *controller* akan mengolah data tersebut sesuai algoritma yang telah ditentukan sebelumnya.

Pengukuran *usage* CPU pada sistem operasi berbasis debian dapat dilakukan dengan berbagai cara. Pada penelitian ini akan menggunakan perintah *ps aux* untuk memonitor penggunaan CPU pada setiap VM yang sedang aktif. Selanjutnya VM akan mengirimkan data *usage* CPU tersebut secara *real time* ke data *base* yang tersimpan pada server *controller*.



Gambar 3.5 Flowchart Pengukuran *Usage* CPU

3.2.4 Prediksi *Usage* CPU Menggunakan *Moving Average*

Metode prediksi yang digunakan adalah metode *moving average*. *Moving average* adalah metode peramalan perataan nilai dengan mengambil sekelompok nilai pengamatan yang kemudian dicari rata-ratanya, lalu menggunakan rata-rata tersebut sebagai ramalan untuk periode berikutnya. Istilah rata-rata bergerak digunakan, karena setiap kali data observasi baru tersedia, maka angka rata-rata yang baru dihitung dan dipergunakan sebagi ramalan.

Prediksi M_t yang diambil adalah Y_t, di mana Y_t adalah jumlah rata-rata dalam lima menit data *usage* CPU VM. Sedangkan dengan *N* adalah 3 periode.

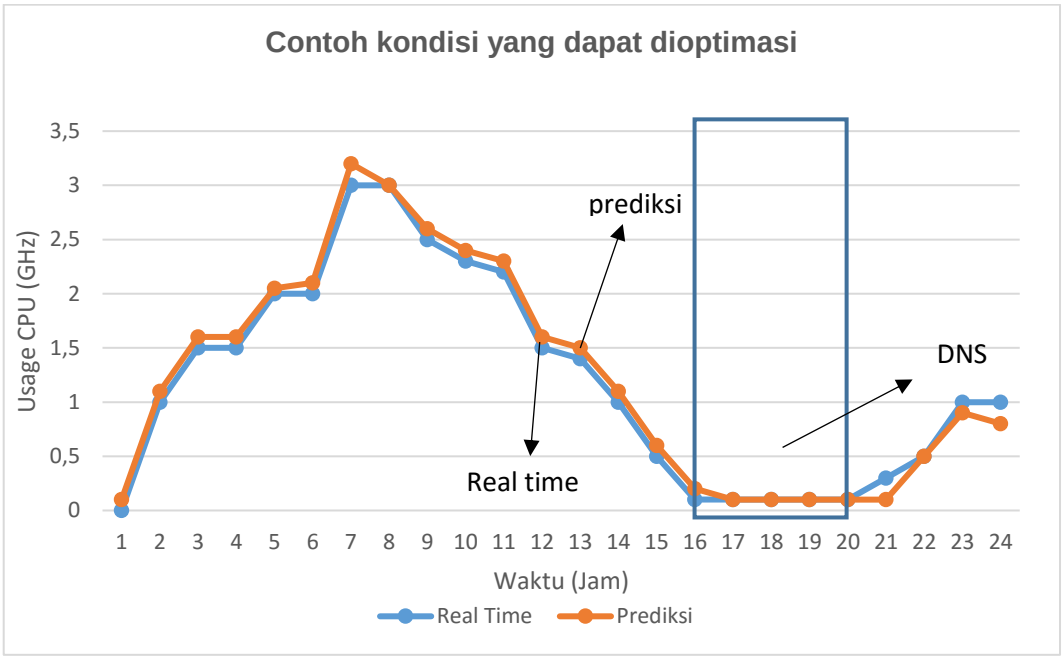
Persamaan matematis *simple moving average* adalah sebagai berikut

$$M_t = F_{t+1} = \frac{Y_t + Y_{t-1} + Y_{t-2} + \cdots + Y_{t-N+1}}{N} \quad (3)$$

Di mana:
M_t adalah *Moving average* untuk periode t
F_{t+1} adalah prediksi *usage* CPU pada periode t + 1
N adalah total jumlah periode rataan.
Y_t adalah nilai *real* pada periode t

3.2.5 Relasi Metode Prediksi dengan *Real Time*

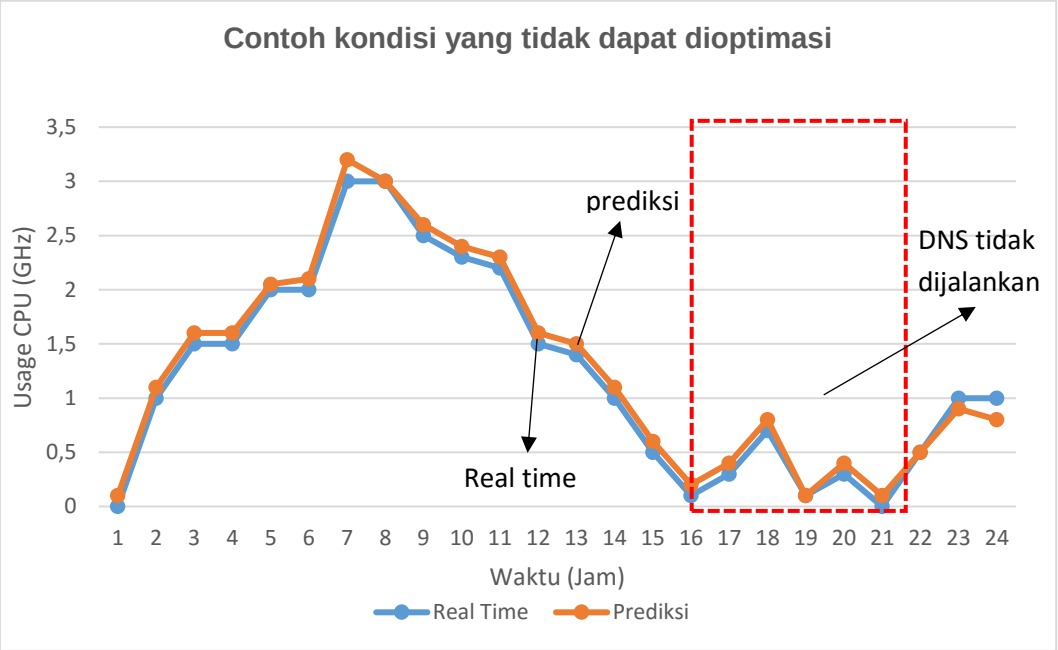
Untuk menunjukkan relasi metode prediksi dengan *real time* dengan lebih jelas, maka akan digunakan contoh grafik kondisi optimasi seperti dalam Gambar 3.6. Dalam Gambar 3.6 terlihat grafik relasi antara prediksi dan *real time* untuk optimasi daya menggunakan DNS. Di mana sumbu x merupakan waktu dalam satuan jam dan sumbu y adalah *usage* CPU dalam bentuk GHz. Dalam grafik terlihat bahwa DNS akan dapat dieksekusi ketika titik prediksi bertemu dengan titik *real time*, eksekusi DNS dalam grafik akan terjadi dalam rentan waktu dari pukul 16 sampai pukul 20, eksekusi DNS ini terjadi dengan mempertimbangkan kondisi seperti ketika titik prediksi berada sesuai dengan titik *real time* dan berada pada kondisi titik *usage* CPU terendah (tanpa beban kerja).



Gambar 3.6 Contoh Relasi Prediksi dan *Real Time* pada Kondisi Daya Dapat Dioptimasi Dengan DNS

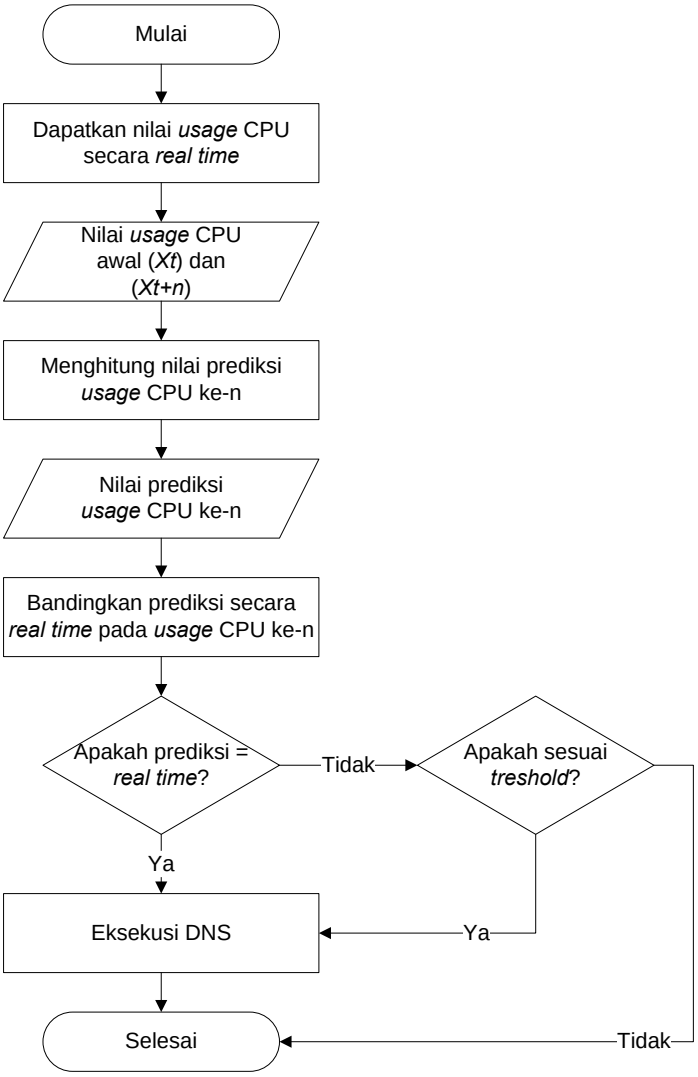
Selanjutnya adalah kondisi dimana relasi prediksi dan *real time* tidak dapat mengoptimasi daya menggunakan skema DNS. Kondisi tersebut terlihat seperti contoh perkiraan pengujian sistem pada Gambar 3.7.

Gambar 3.7 merupakan contoh gambaran yang menunjukkan di mana relasi prediksi dan *real time* memutuskan optimasi daya pada data *center* tidak dapat dilakukan. Sebenarnya pada kondisi ini optimasi masih dapat dilakukan, namun jika optimasi tetap dilakukan maka kemungkinan besar akan berpengaruh pada performa sistem dengan meningkatnya *response time*. Hal ini bisa dilihat pada bagian DNS yaitu dalam rentan waktu dari pukul 16 sampai pukul 22 tidak dapat dieksekusi walaupun *real time* pada pukul 16 telah menunjukkan bahwa *usage* CPU berada pada kondisi titik terendah, namun karena pada pukul 17 dan seterusnya diprediksikan *usage* CPU akan terus meningkat.



Gambar 3.7 Contoh Relasi Prediksi dan *Real Time* terhadap Kondisi Daya Yang Tidak Dapat Dioptimasi

Dari hasil penjelasan relasi prediksi dan *real time* di atas maka dapat dibuatkan *flowchart* sederhana seperti berikut ini.



Gambar 3.8 Relasi Prediksi dan *Real Time*

3.2.6 Penanganan terhadap Kegagalan Prediksi

Sebenarnya kegagalan prediksi adalah sebuah hal yang paling dihindari dalam penelitian ini, sehingga akan diupayakan untuk menerapkan model dengan metode yang tepat agar prediksi benar-benar akurat, namun dalam hal prediksi tentunya ada kemungkinan muncul kegagalan prediksi, kegalan prediksi dalam sistem akan mempengaruhi pula performa sistem keseluruhan, oleh karena itu perlu adanya mekanisme untuk mencegah hal tersebut dapat terjadi.

Kegagalan prediksi yang akan menimbulkan efek pada performa dalam penelitian ini adalah pada saat terjadinya eksekusi DNS, karena server yang telah mengalami eksekusi DNS akan membutuhkan waktu lama untuk normal kembali.

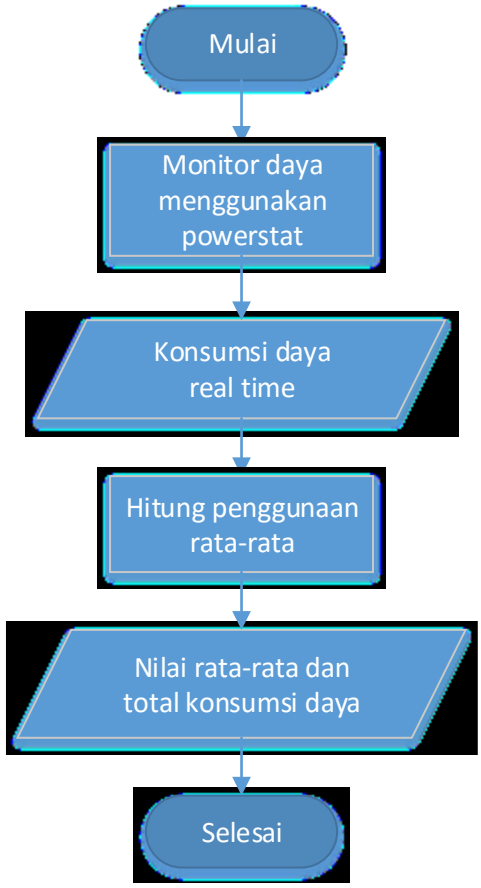
Penanganan kegagalan eksekusi DNS dapat dilakukan dengan membuat cadangan server. Artinya tidak semua server dalam keadaan *workload* rendah dieksekusi dengan DNS. Sistem akan menyisakan beberapa cadangan server untuk mengantisipasi kemungkinan *workload* kembali meningkat akibat kesalahan prediksi. Sehingga jika ada 70% server dari jumlah keseluruhan server data *center* yang dapat dioptimasi dengan DNS, maka hanya 60% - 65% server saja yang akan dioptimasi atau dieksekusi dengan DNS, sisanya 5% - 10% akan *standby* untuk mencegah penurunan performa akibat kegagalan prediksi. Sehingga dalam kondisi ini data *center* tetap dapat penanganan optimasi daya secara maksimal.

3.2.7 Pengukuran Konsumsi Daya

Konsumsi daya merupakan hal yang menjadi tujuan dalam penelitian ini, semua proses ataupun mekanisme yang terdapat pada sistem yang dibangun merupakan upaya untuk menekan konsumsi daya sampai titik minimal.

Konsumsi daya dalam uji coba sistem pada penelitian ini akan memanfaatkan *tool* dari keluarga Linux Debian yaitu Powerstat dengan menggunakan basis ACPI (*Advanced Configuration and Power Interface*) (Hewlett-Packard et al., 2011), merupakan sebuah spesifikasi industri yang terbuka yang mengizinkan para desainer perangkat lunak untuk mengintegrasikan fitur-fitur manajemen daya dalam sebuah sistem komputer, yang mencakup perangkat keras, sistem operasi, dan perangkat lunak aplikasi.

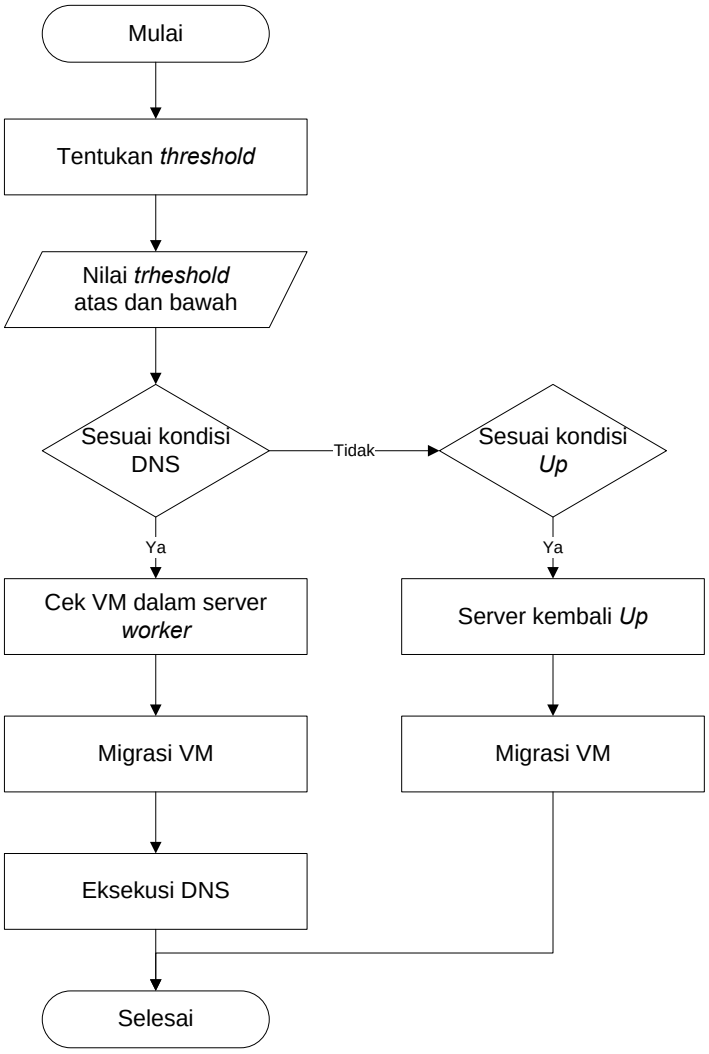
Pengukuran konsumsi daya dilakukan pada server *worker* 1 dan *worker* 2 sesuai durasi uji coba sistem. Daya yang yang diukur adalah sesuai kondisi *usage* CPU secara *real time*. Selanjutnya akan dihitung nilai rata-rata dari konsumsi daya setiap server *worker*, kemudian akan dibandingkan antara konsumsi daya dengan pengujian yang berbeda.



Gambar 3.9 Pengukuran Konsumsi Daya

3.2.8 Optimasi Daya dengan *Dynamics Shutdown* (DNS)

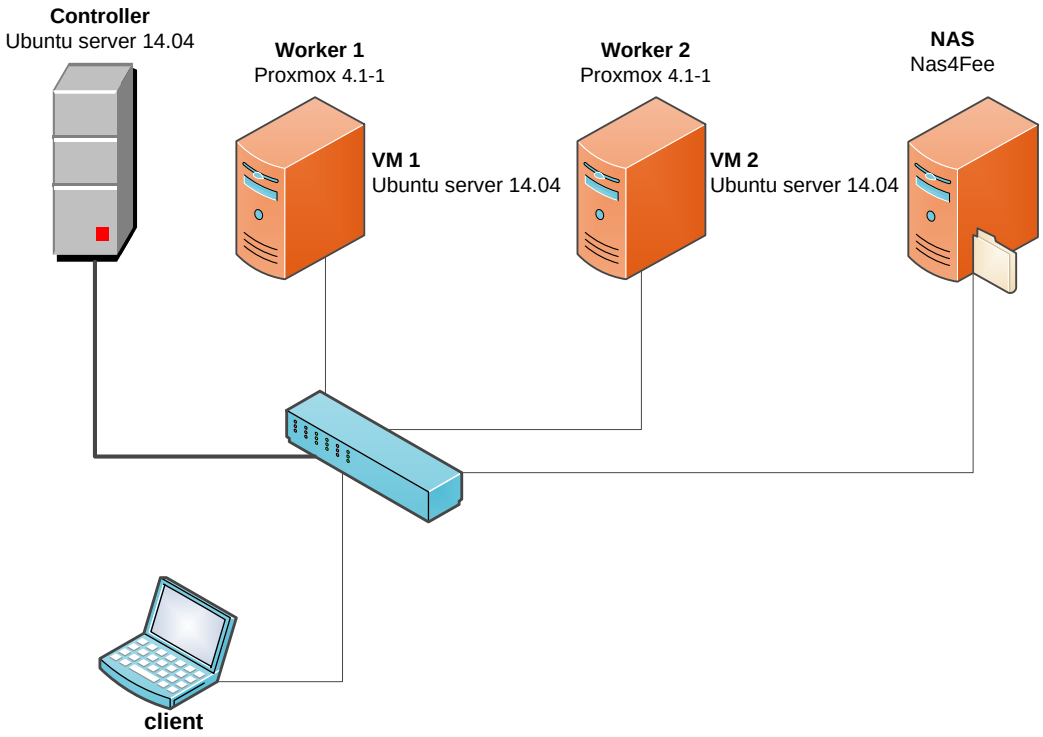
Optimasi daya yang akan dilakukan pada penelitian ini adalah dengan mempertimbangkan performa sistem tetap berada pada keaadaan optimal, Sehingga penghematan konsumsi daya pada server data *center* yang dilakukan pada DNS dapat dilakukan dengan bantuan prediksi *usage* CPU seperti yang telah dijelaskan sebelumnya. Eksekusi DNS akan bekerja sesuai nilai *threshold* yang telah ditentukan. Nilai *threshold* akan ditentukan setelah evaluasi server, nantinya akan dipertimbangkan bagaimana *usage* CPU pada kondisi tanpa beban dan kondisi *usage* CPU dengan beban tinggi. Sebelum DNS dijalankan, maka sistem akan menjalankan migrasi VM ke server *worker* yang yang lain terlebih dahulu. Selanjutnya server kembali dalam kondisi *Up* ketika *threshold* DNS sudah tidak memenuhi lagi.



Gambar 3.10 Optimasi Daya dengan DNS

3.3 Rancangan Sistem Uji Coba

Pelaksanaan uji coba sistem dilakukan pada virtual komputer dengan memanfaatkan VMware Workstation. Ada beberapa komputer server virtual yang dipersiapkan dalam VMware Workstation, di antaranya adalah satu komputer sebagai server *controller* dan dua komputer sebagai server *worker*, satu server sebagai NAS serta ada satu komputer *client.*



Gambar 3.11 Arsitektur *Cloud Computing* sebagai Pengujian Sistem

1. **Server *Worker***

   Server *worker* akan diinstal Linux Proxmox sebagai sistem operasi sekaligus sebagai *framework cloud computing.* Setiap server *worker* akan diinstal satu server VM dengan sistem operasi Linux Ubuntu server 14.04. Semua *worker* juga akan dibuatkan program yang dapat membaca kondisi server yaitu membaca *workload* penggunaan CPU (*usage* CPU), list jumlah VM yang aktif. Program tersebut akan mengirim data secara *real time* ke server *controller*.

2. **VM (*Virtual Machine*)**

   Server VM adalah server yang berada dalam *cloud computing* (server Proxmox). Server VM dapat berpindah (migrasi) dari server *worker* 1 ke server *worker* 2 ataupun sebaliknya sesuai intruksi penjadwalan yang telah dibuat oleh server *controller*.

   Pada server VM juga ditanam program dalam bahasa *bash script* yang bertujuan untuk membaca kondisi server VM itu sendiri dan mengirimkan ke *controller*

yang akan diolah sebagai data *forecast*. Data tersebut akan dikirimkan setiap menit.

3. Server *Controller*

Server *controller* adalah sebagai pengatur yang akan bertindak untuk memanajemen sistem yang dibangun. Pada server ini akan dibuat program untuk menerima kiriman data *usage* oleh *worker*, dan VM. Selanjutnya program tersebut akan mengolah kiriman data dengan prediksi sesuai algoritma *moving average* yang telah ditetapkan dan mempertimbangkan syarat-syarat DNS seperti yang telah dijelaskan di atas secara *real time*. Selanjutnya *controller* akan mengirimkan perintah pada *worker* yang berupa migrasi VM, eksekusi DNS, atau *Up* kembali. Server *controller* yang akan memutuskan *worker* mana yang dioptimasi daya dengan DNS. Pada *controller* juga akan menampilkan kondisi dari semua server, di antaranya *usage* CPU, konsumsi daya, *Up time*, jadwal migrasi atau DNS.

4. NAS

NAS yang digunakan pada uji coba penelitian ini adalah dengan sistem operasi Nas4Free. Penggunaan NAS bertujuan untuk memudahkan migrasi VM antar server *worker*.

5. *Client*

Komputer *client* akan diinstal aplikasi Httperf untuk membangkitkan data dan mengirim/mengakses ke komputer server VM yang akan mengakibatkan fluktuasi *usage* CPU pada server VM dan *worker*. Pada komputer *client* juga akan digunakan untuk memantau ketersedian sistem yang dibangun. Parameter yang akan dilihat untuk memeriksa ketersedian sistem adalah *response time*.

Sesuai penjelasan sebelumnya, adapun parameter pengujian yang akan digunakan untuk optimasi daya adalah *usage* CPU atau frekuensi dari CPU dalam satuan persen. Untuk evaluasi ketersedian sistem adalah mengetahui tingkat performa sistem dengan menggunakan parameter seperti *response time* yang terjadi

pada server *cloud computing*. Sedangkan untuk estimasi konsumsi daya menggunakan *tool* Linux Powerstat yang terdapat beberapa parameter di antaranya adalah *usage* CPU, ACPI, *memory* RAM, I/O

3.3.1 Perangkat Keras dan Perangkat Lunak yang Digunakan

Perangkat keras yang digunakan adalah sebuah laptop sebagai tempat berjalannya *virtual machine* VMware Workstation 12. Adapun spesifikasinya adalah sebagai berikut:

1. Prosesor 2.5-GHz Intel Core i5-2520M
2. Memory RAM 8 Gb
3. Hardisk 300 Gb
4. Sistem opersi Windows 10

Perangkat lunak yang digunakan pada penelitian ini adalah:

1. VMware Workstation 12 sebagai virtualisasi server
2. Proxmox versi 4.1-1 sebagai basis *cloud computing*
3. Ubuntu server 14.04 sebagai sistem operasi yang terinstal pada server *controller* dan server VM
4. Nas4Free yang digunakan untuk data *storage* VM *cloud computing*, berguna agar migrasi dapat berjalan dengan lancar
5. Apache server sebagai *web* server pada server *controller* untuk manajemen sistem dan *web* server pada VM sebagai uji coba penerima beban
6. Mysql merupakan database yang digunakan untuk menampung semua data server yang terhubung dengan sistem
7. PHP merupakan bahasa pemrograman yang digunakan untuk menjalankan sistem, baik sebagai kalkulasi, *forecast moving average*, dan penjadwalan
8. Powerstat adalah perangkat lunak yang digunakan untuk memantau konsumsi daya pada server *worker*
9. Httperf sebagai pembangkit beban.

3.3.2 Pengukuran *Usage* CPU

Usage CPU yang dipantau adalah dalam satuan persen. Pengukuran *usage* menggunakan perintah *ps aux*. Hasil dari perintah *ps aux* akan menampilkan terlalu banyak informasi, seperti *usage* CPU, penggunaan *memory*, waktu mulai aplikasi, user dan *daemon*. Namun hanya informasi yang dibutuhkan saja yang akan diambil.

Untuk mengambil nilai dari *usage* CPU dalam persen saja, maka dapat menggunakan perintah seperti di bawah ini.

```
ps aux | awk {'sum+=$3;print sum'} | tail -n 1
```

Pengambilan informasi *usage* CPU akan dilakukan secara otomatis dengan memanfaatkan *crontab* Linux setiap 10 detik sekali pada server VM. Data *usage* tersebut akan dikirim secara *real time* munuju server *controller*. Pada server *controller* akan dihitung rata-rata *usage* CPU perlima menit. Hal ini bertujuan agar prediksi yang dilakukan adalah berdasarkan lima menit.

3.3.3 Prediksi Menggunakan *Moving Average*

Script moving average akan dibuat dengan menggunakan pemrograman PHP dan berada pada server *controller*. Di mana akan mengolah setiap data yang dikirimkan oleh server VM dan *worker*.

Moving average akan bekerja secara *real time*, setiap data *usage* CPU yang masuk akan langsung dihitung untuk memprediksi nilai berikutnya. Nilai *usage* CPU yang dihitung adalah nilai rata-rata perlima menit, dan selanjutnya akan diprediksi nilai *usage* CPU pada lima menit berikutnya.

Nilai *usage* CPU yang dijadikan sebagai data *forecast* atau prediksi adalah nilai *usage* CPU dari server VM, bukan nilai *usage* CPU server *worker*. Hal ini dikarenakan *usage* CPU dari server VM tidak mengalami perlakuan *shutdown*, sehingga nilai *usage* CPU dapat dikirimkan secara terus menurus tanpa henti walaupun server *worker* telah dimatikan.

3.3.4 Pengukuran Konsumsi Daya

Dengan memanfaatkan ACPI (Advanced Configuration and Power Interface), maka pengukuran konsumsi daya pada server *cloud computing* dapat menggunakan *tool* Powerstat yang tersedia dalam lingkungan Linux Debian. Karena Proxmox merupakan Linux yang berbasis debian, maka pada penelitian ini cukup menggunakan Powerstat dalam pemantauan konsumsi daya. Pengukuran konsumsi daya dengan *tool* Powerstat harus menggunakan baterai dan tidak boleh melalukan *charge* pada laptop pengujian.

Pengukuran konsumsi daya dilakukan pada server *worker*1 (Proxmox1) dan *worker*2 (Proxmox2) yang telah diinstal sistem operasi Proxmox. Daya yang diukur adalah sesuai kondisi *usage* CPU secara *real time* dalam satuan Watt. Selanjutnya akan dihitung nilai rata-rata dari konsumsi daya setiap server *worker* tersebut.

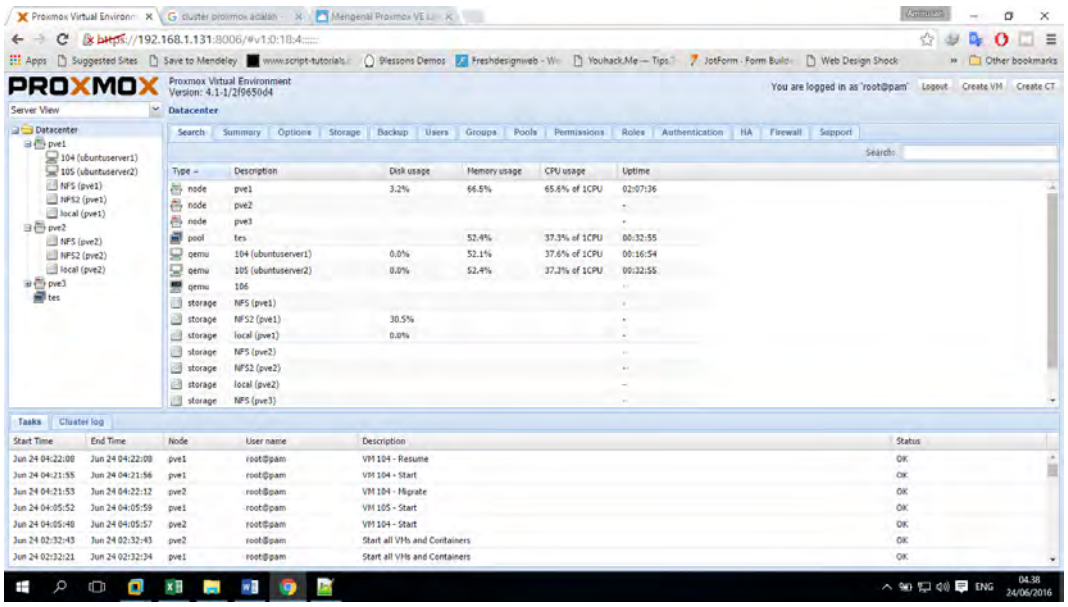
BAB 4
UJI COBA DAN EVALUASI

4.1 Implementasi Sistem

4.1.1 Implementasi Sistem *Cloud computing*

Sistem *cloud computing* yang dibangun menggunakan Proxmox VE (Virtual Environtment) versi 4.1-1 yang merupakan distribusi berbasis Debian etch (x86_64). Proxmox adalah *platform* virtualisasi bersifat *open source* yang mendukung untuk menjalankan *virtual* mesin berbasis KVM dan OpenVZ.

Ada dua server Proxmox yang digunakan yaitu server *worker* 1 dan server *worker* 2. Kedua server Proxmox ini menjadi sebuah *cluster cloud computing*. Selanjutnya dalam cluster tersebut terdapat dua server VM dengan sistem operasi Ubuntu server. Berikut adalah tampilan antarmuka sistem *cloud computing* berbasis Proxmox yang telah dibangun.



Gambar 4.1 Antarmuka *Cloud* Proxmox VE

4.1.2 Implementasi Aplikasi pada *Controller*

Dalam uji coba penelitian ini, server *controller* bertugas sebagai pengatur semua server yang berjalan pada sistem *cloud computing* yang telah dibangun. Server *controller* akan menghitung nilai prediksi, memutuskan jadwal migrasi dan DNS atau *shutdown* server.

Ada beberapa *script* progam yang terdapat pada sisi *controller*, di antaranya adalah

1. Forecast_vm.php

   Script ini akan berjalan secara *real time* untuk melakukan prediksi dengan mengolah nilai *real* yang telah dikirimkan oleh server VM. Akan dihitung nilai rata-rata *real* terlebih dahulu selama 5 menit, kemudian akan diprediksi nilai rata-rata menit berikutnya.
2. Akumulasi_data.php

   Script ini bertugas menggabungkan informasi antara VM dan *worker* dan mengupdate data keberadaan VM di worker
3. Generate_schedule.php

   Semua jadwal untuk eksekusi migrasi VM, *shutdown* server dan *up* server akan dieksekusi dengan *script* ini.
4. Eksekusi_mirate.php

   Pada *script* ini akan dilakukan migrasi server sesuai jadwal yang telah diberikan oleh *script* generate_schedule.php
5. Eksekusi_shutdown.php

   Sama halnya dengan *script* eksekusi_migrate.php, namun yang berbeda pada *script* ini hanyalah bertugas untuk *shutdown* server.
6. Eksekusi_up.php

   Up server akan dilakukan dengan menghidupkan server worker dengan perintah *power on* pada VMware dengan bantuan *script* ini secara otomatis sesuai jadwal.

Selain *script* utama yang telah dijelaskan di atas, sebenarnya ada beberapa *script* lagi yang terdapat pada server *controller.*

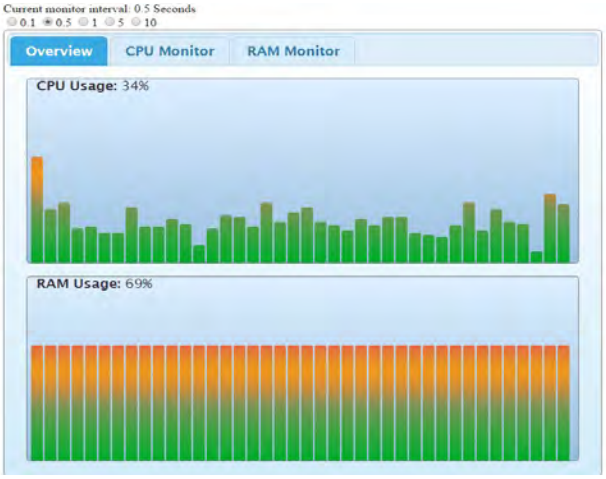
4.1.3 Implementasi Monitor VM

Data yang dibutuhkan dari server VM, seperti *usage* CPU dalam satuan persen akan dikirim otomatis dari server VM ke server *controller* setiap menit. Selanjutnya data tersebut akan ditampung dalam database mysql pada server *controller*. Pengiriman dan pengambilan data *usage* CPU dari server VM ke server *controller* dilakukan dengan menjalankan file pemrograman *bash script* Linux, di mana pada file *bash script* tersebut berisi perintah seperti potongan *script* berikut:

```
/sbin/ifconfig eth0 | grep "inet addr" | awk -F: '{print $2}' | awk
'{print $1}'
cat /sys/class/net/eth0/address
ps aux | awk {'sum+=$3;print sum'} | tail -n 1
```

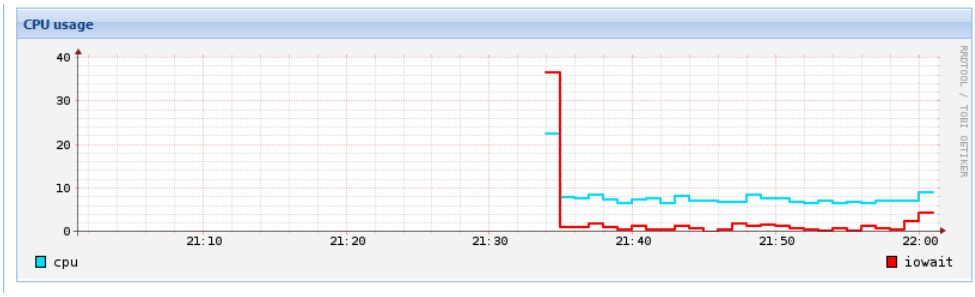
Dari *script* di atas terlihat ada tiga informasi penting dari server VM yang akan dikirimkan ke server *controller* yaitu IP address, mac address, dan nilai *usage* CPU.

Selain itu, pemantauan server VM juga dapat dilakukan menggunakan grafik *interface* yang dapat diakses langsung menggunakan *web browser* seperti terlihat dalam Gambar 4.2.



Gambar 4.2 Grafik Monitor Server VM

Selain dengan grafik seperti yang ditunjukkan pada Gambar 4.2, sebagai antar muka sistem dapat dipantau langsung melalui Proxmox seperti diperlihatkan dalam Gambar 4.3.



Gambar 4.3 Pemantauan *usage* CPU Menggunakan Proxmox

4.2 Skenario Uji Coba

Adapun skenario uji coba pada penelitian ini meliputi:

1. Distribusi Beban

   Distribusi beban dilakukan sebagai tahap awal pengujian pada penelitian ini, hal ini bertujuan untuk menghasilkan trafik yang dapat mempengaruhi *usage* CPU. Ada dua pola pengujian yang akan dihasilkan, yaitu pola beban meningkat dan pola beban menurun. Selanjutnya akan dilihat baagaimana pengaruh trafik *request* terhadap *usage* CPU

2. Uji Coba Sistem Menggunakan Metode Prediksi

   Pada pengujian ini terdapat beberapa sub pengujian yang akan dilakukan adalah:

   1. Uji Coba Akurasi Metode pada Beban Meningkat

      Uji coba ini bertujuan untuk melihat bagaimana tingkat akurasi metode prediksi yang digunakan pada beban meningkat.

   2. Uji Coba Akurasi Metode pada Beban Menurun

      Uji coba ini bertujuan untuk melihat bagaimana tingkat akurasi metode prediksi yang digunakan pada beban menurun.

   3. Uji Coba Konsumsi Daya pada Beban Meningkat

      Uji coba akan dilakukan dengan melihat nilai konsumsi daya, selanjutnya akan dilihat bagaimana *usage* CPU dapat mempengaruhi konsumsi daya pada sistem.

4. Uji Coba Konsumsi Daya pada Beban Menurun

   Uji coba bertujuan untuk melihat bagaimana pengaruh *usage* CPU terhadap konsumsi daya pada sistem.

5. Uji Coba *Response Time* pada Beban Meningkat

   Langkah uji coba ini dimaksudkan untuk mengetahui berapa jumlah nilai *response time* yang terjadi pada beban meningkat.

6. Uji Coba *Response Time* pada Beban Menurun

   Langkah uji coba ini dimaksudkan untuk mengetahui berapa jumlah nilai *response time* yang terjadi pada beban menurun.

3. Uji Coba dengan Metode Konvensional

   Sama seperti uji coba pada sistem yang menggunakan metode prediksi, uji copa dengan metode konvensional juga akan dilakukan dengan beberapa sub pengujian, yaitu:

   1. Uji Coba Konsumsi Daya pada Beban Meningkat
   2. Uji Coba Konsumsi Daya pada Beban Menurun
   3. Uji Coba *Response Time* pada Beban Meningkat
   4. Uji Coba *Response Time* pada Beban Menurun

4.3 Langkah Uji Coba

Langkah uji coba pada penelitian ini dapat dirumuskan menjadi beberapa tahap seperti berikut ini.

1. Langkah awal adalah menentukan nilai distribusi trafik yang dibangkitkan dengan model distribusi Poisson. Hal ini bertujuan agar distribusi trafik dapat bernilai acak.
2. Data beban kerja dalam bentuk *request* http akan dibangkitkan dengan menggunakan Httperf melalui komputer *client*, yang akan mengakibatkan *usage* CPU meningkat sesuai jumlah *request*.
3. Server VM dan Server *worker* akan mengirimkan data *usage* CPU ke *controller* secara berkala setiap 10 detik.
4. Program pada server *controller* akan membaca data yang dikirimkan oleh server *worker* dan VM untuk memprediksi *usage* CPU.

5. *Controller* akan memutuskan jadwal optimasi daya dengan migrasi dan DNS.
6. *Controller* akan menghitung estimasi penghematan daya dengan *tool* Powerstat.

Performa kinerja data *center* akan dilihat melalui komputer *client* dengan memperhitungkan nilai *response time* dan *latency* server serta mendeteksi ada tidaknya server yang *overload*.

4.4 Hasil dan Uji Coba

Pada sub-bab hasil dan uji coba ini dipaparkan hasil dan uji coba dari sistem. Secara garis besar pengujian akan dilakukan pada dua model skenario sistem, yaitu skenario pertama adalah dengan menggunakan metode prediksi dan pada skenario kedua adalah dengan metode konvensional. Sistem dengan model prediksi adalah sistem yang dibangun dengan penambahan metode *moving average,* di mana sistem dapat melakukan migrasi VM dan *shutdown* server secara otomatis. Sedangkan sistem yang dibangun dengan model konvensional di sini artinya adalah sistem *cloud* yang dibangun seperti pada umumnya, di mana sistem tidak menggunakan metode tambahan, tidak melakukan migrasi ataupun *shutdown* server secara otomatis.

Hasil uji coba akan dibahas dengan menampilkan nilai *response time* dan konsumsi daya pada saat peningkatan trafik dan pada saat penurunan trafik. Pengujian *response time* dilakukan untuk melihat bagaimana performa sistem *cloud* yang telah dibangun, sedangkan pengujian konsumsi daya bertujuan untuk melihat seberapa besar penghematan konsumsi daya yang dapat diterapkan.

4.4.1 Distribusi Beban

Distribusi beban pada pengujian penelitian ini adalah memberikan beban berupa *request* http ke sever VM1 dan VM2, sehingga nantinya akan berpengaruh terhadap *usage* CPU dari kedua server VM tersebut. Data *usage* CPU akan dikirimkan otomatis ke server *controller*.

Sebenarnya pengujian yang akan direncanakan adalah selama satu hari yaitu 12 (dua belas) jam dan data *usage* CPU yang diambil adalah rata-rata dalam satu jam, sehingga sistem dapat lebih efektif dalam menghitung nilai prediksi perjam. Pengujian selama datu hari lebih menunjukkan sistem yang sebenarnya. Namun

karena keterbatasan waktu dan perangkat pengujian, maka pengujian diminimalkan menjadi hanya satu jam saja, di mana data *usage* CPU yang diambil adalah rata-rata dalam 5 menit, dan akan diprediksi nilai rata-rata *usage* CPU untuk 5 menit berikutnya. Pengujian akan dilakukan dari menit ke-0 sampai menit ke-55, sehingga total nilai *usage* CPU berjumlah 12 sampel, dan ini sama halnya seperti pengujian yang dilakukan dalam satu hari dengan durasi 12 jam. Selanjutnya sampel dalam pengujian akan ditentukan dengan pemodelan trafik distribusi Poisson. Distribusi beban dengan durasi 1 jam tersebut akan dibangkitkan untuk menghasilkan beban naik dan beban turun.

Distribusi beban pada pengujian ini dibangkitkan dengan mengenerate trafik secara berkala dan terus menerus dengan bantuan *tool* Httperf pada lingkungan Linux. Namun sebelum trafik dibangkitkan dengan *tool* Httperf, maka terlebih dahulu akan ditentukan pemodelan trafik dengan mengunakan distribusi Poisson. Nilai trafik distribusi Poisson ini didapatkan dengan generate menggunakan Microsoft Excel dengan fungsi POISSON.DIST(X;MEAN;FALSE).

Tabel 4.1 Data Pemodelan Trafik dengan Distribusi Poisson

No.	**Time**	**X**	**Probabilitas**		**Trafik**	**Trafik Perdetik (request)**	**Jumlah Trafik dalam 5 menit (request)**
	Menit		**mean**	**7,0**	**x 950**		
1	00	0	0,0009		0,866	1	300
2	05	1	0,0064		6,064	6	1800
3	10	2	0,0223		21,224	21	6300
4	15	3	0,0521		49,523	50	15000
5	20	4	0,0912		86,665	87	26100
6	25	5	0,1277		121,331	121	36300
7	30	6	0,1490		141,553	142	42600
8	35	7	0,1490		141,553	142	42600
9	40	8	0,1304		123,859	124	37200
10	45	9	0,1014		96,334	96	28800
11	50	10	0,0710		67,434	67	20100
12	55	11	0,0452		42,913	43	12900

Nilai X dalam Tabel 4.1 terlihat berjumlah 12 sampel dimulai dari 0 sampai dengan 11, hal ini didasari karena jumlah sampel dari menit ke-00 sampai dengan menit ke-55 berjumlah 12 sampel. Selanjutnya nilai *mean* yang ditentukan adalah 7,0 karena dalam distribusi beban tersebut mengalami kenaikan dan penurunan

selama pengujian berlangsung satu jam. Nilai 7 juga mengindentifikasi bahwa kenaikan dengan nilai tertinggi akan terjadi pada sampel ke-7 yaitu kira-kira pada menit ke-30.

Dalam Tabel 4.1 terlihat bahwa nilai-nilai probabilitas yang dihasilkan bernilai di bawah 1, oleh karena itu nilai tersebut harus dinaikkan untuk mengasilkan nilai *request* yang cukup untuk membangkitkan beban *request* dengan Httperf. Angka 950 dipilih sebagai perkalian dengan nilai probabilitas yang dihasilkan sebelumnya, hal ini disebabkan karena dengan perkalian nilai 950 menghasilkan *request* maksimal yang dirasa cukup, yaitu menghasilkan *request* maksimal 141,553. Angka ini dinilai maksimal karena berdasarkan analisa dan pengujian sebelumnya, dengan *request* rata-rata 140 perdetik pada sistem pengujian akan menaikkan *usage* CPU hingga rata-rata 30%. Nilai batas rata-rata 30 % dipilih karena batas tersebut yang lebih memungkinkan dalam pengujian. Jika satu VM mengambil *resource* CPU 30% maka untuk 2 VM akan menghabiskan 60% *resource* CPU dan ditambah server *controller* serta server *worker*, maka setelah menganalisa dari beberapa uji coba, nilai ini dirasa cukup. Sebenarnya hal ini lebih disebabkan karena keterbatasan *resource* laptop yang dimiliki saat pengujian. Ada beberapa server virtual yang berjalan sekaligus dalam satu laptop, jika *usage* server VM tersebut melebihi rata-rata 30% hal ini dapat menguras *resource* laptop terlalu banyak yang berakibat gagalnya pengujian akibat laptop mengalami *hang*.

Nilai probabilitas yang telah dihasilkan harus dibulatkan agar dapat diimplementasi dengan *tool* Httperf seperti yang terlihat dari kolom trafik perdetik dalam Tabel 4.1 Selanjutnya setiap sampel akan ditentukan total jumlah *request* selama lima menit.

Dalam Tabel 4.1 terlihat bahwa ada 12 sampel *request* ataupun koneksi, masing-masing sampel *request* akan berlangsung selama 5 menit, dimulai dari menit ke-00 smpai dengan menit ke-55. Nilai *request* perdetik setiap lima menit awal akan berbeda dengan nilai *request* perdetik pada lima menit kedua, dan begitu seterusnya. Sebagai contoh pada menit ke-00, *request* akan berlangsung dari menit ke-00 sampai menit ke-05 dan memiliki nilai 1 *request* perdetik, sehingga total *request* yang dijalankan selama lima menit tersebut adalah 300 *request*.

Setelah pemodelan trafik didapatkan dengan distribusi Poisson, maka nilai-nilai tersebut akan dibangkitkan dengan *tool* Httperf agar dapat menghasilkan trafik pengujian. Trafik akan dibangkitkan sesuai jadwal masing-masing dengan memanfaatkan *crontab* Linux.

```
0 * * * *   httperf --server=192.168.1.175 --uri=/index.php --num-conns=300 --rate=1
5 * * * *   httperf --server=192.168.1.175 --uri=/index.php --num-conns=1800 --rate=6
10 * * * * httperf --server=192.168.1.175 --uri=/index.php --num-conns=6300 --rate=21
15 * * * * httperf --server=192.168.1.175 --uri=/index.php --num-conns=15000 --rate=50
20 * * * * httperf --server=192.168.1.175 --uri=/index.php --num-conns=26100 --rate=87
25 * * * * httperf --server=192.168.1.175 --uri=/index.php --num-conns=36300 --rate=121
30 * * * * httperf --server=192.168.1.175 --uri=/index.php --num-conns=42600 --rate=142
35 * * * * httperf --server=192.168.1.175 --uri=/index.php --num-conns=42600 --rate=142
40 * * * * httperf --server=192.168.1.175 --uri=/index.php --num-conns=37200 --rate=124
45 * * * * httperf --server=192.168.1.175 --uri=/index.php --num-conns=28800 --rate=96
50 * * * * httperf --server=192.168.1.175 --uri=/index.php --num-conns=20100 --rate=67
55 * * * * httperf --server=192.168.1.175 --uri=/index.php --num-conns=12900 --rate=43

0 * * * *   httperf --server=192.168.1.176 --uri=/index.php --num-conns=300 --rate=1
5 * * * *   httperf --server=192.168.1.176 --uri=/index.php --num-conns=1800 --rate=6
10 * * * * httperf --server=192.168.1.176 --uri=/index.php --num-conns=6300 --rate=21
15 * * * * httperf --server=192.168.1.176 --uri=/index.php --num-conns=15000 --rate=50
20 * * * * httperf --server=192.168.1.176 --uri=/index.php --num-conns=26100 --rate=87
25 * * * * httperf --server=192.168.1.176 --uri=/index.php --num-conns=36300 --rate=121
30 * * * * httperf --server=192.168.1.176 --uri=/index.php --num-conns=42600 --rate=142
35 * * * * httperf --server=192.168.1.176 --uri=/index.php --num-conns=42600 --rate=142
40 * * * * httperf --server=192.168.1.176 --uri=/index.php --num-conns=37200 --rate=124
45 * * * * httperf --server=192.168.1.176 --uri=/index.php --num-conns=28800 --rate=96
50 * * * * httperf --server=192.168.1.176 --uri=/index.php --num-conns=20100 --rate=67
55 * * * * httperf --server=192.168.1.176 --uri=/index.php --num-conns=12900 --rate=43
```

Seperti *script* yang terlihat di atas, jadwal generate trafik akan berlangsung setiap lima menit dari menit ke-00 sampai dengan menit ke-55. *Request* ataupun koneksi akan dilakukan pada dua server VM yang beralamat di 192.168.1.175 dan 192.168.1.176. *Request* yang diberikan adalah berupa http yang akan mengakses halaman index.php sesuai jumlah yang telah ditentukan. Selanjutnya akan dipantau nilai *usage* CPU yang telah dipengaruhi oleh distribusi beban melalui *request* http index.php tersebut.

Tabel 4.2 *Usage* CPU pada VM1 selama 1 Jam Pengujian

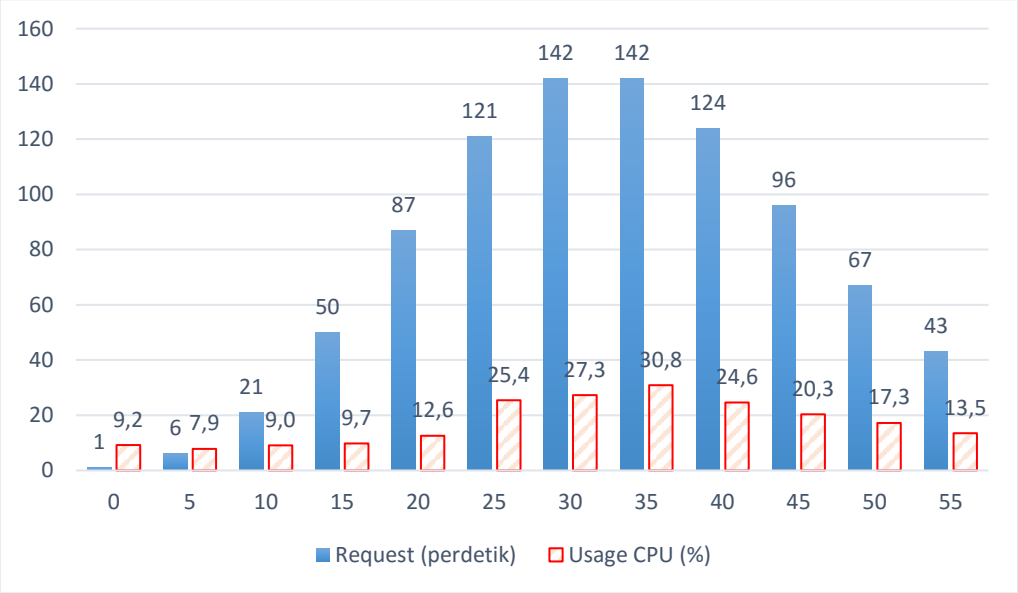
No	Menit	Request perdetik	Rata -rata Usage CPU dalam 5 menit (%)
1	00	1	9.2
2	05	6	7.9
3	10	21	9.0
4	15	50	9.7
5	20	87	12.6
6	25	121	25.4
7	30	142	27.3
8	35	142	30.8
9	40	124	24.6
10	45	96	20.3
11	50	67	17.3
12	55	43	13.5

Tabel 4.3 *Usage* CPU pada VM2 selama 1 Jam Pengujian

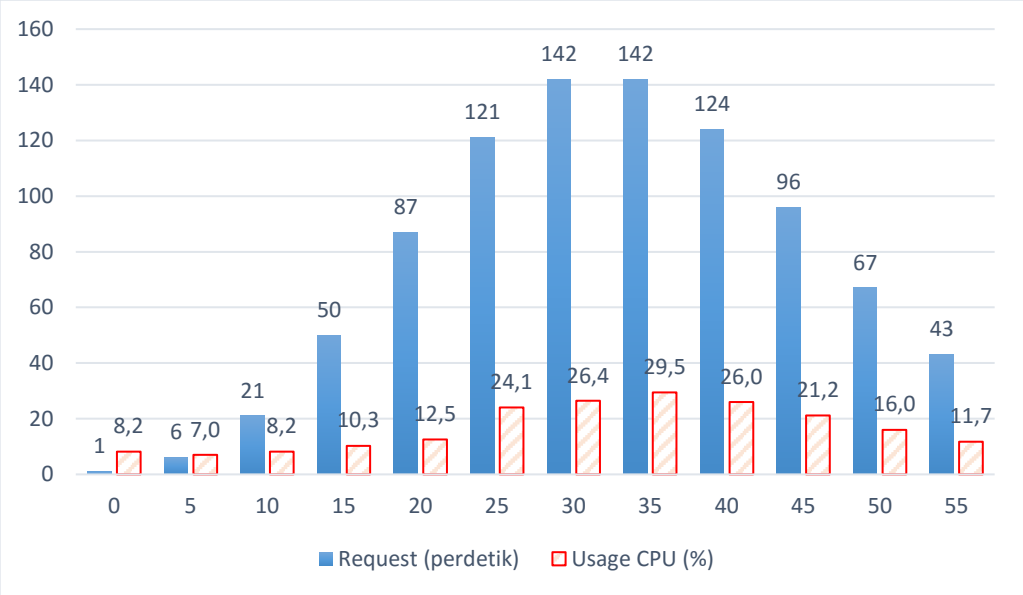
No	Menit	Request perdetik	Rata -rata Usage CPU dalam 5 menit (%)
1	00	1	8.2
2	05	6	7.0
3	10	21	8.2
4	15	50	10.3
5	20	87	12.5
6	25	121	24.1
7	30	142	26.4
8	35	142	29.5
9	40	124	26.0
10	45	96	21.2
11	50	67	16.0
12	55	43	11.7

Dalam Tabel 4.2 dan Tabel 4.3 merupakan hasil uji coba berdasarkan model trafik distribusi Poisson yang telah dibahas sebelumnya, terlihat perbandingan antara jumlah *request* yang diberikan dan rata-rata *usage* CPU yang dihasilkan.

Semakin tinggi nilai *request* maka semakin tinggi pula nilai rata-rata *usage* CPU. Jika disajikan dengan grafik maka akan tampil seperti Gambar 4.4 dan Gambar 4.5.



Gambar 4.4 Grafik Hubungan *Request* dengan *Usage* CPU pada VM1



Gambar 4.5 Grafik Hubungan *Request* dengan *Usage* CPU pada VM2

Dari Gambar 4.4 dan Gambar 4.5 menunjukkan grafik hubungan antara jumlah *request* dengan nilai *usage* CPU. Dapat diperhatikan bahwa nilai *usage* CPU yang dihasilkan dengan jumlah *request* yang diberikan sangat berbeda jauh. Namun pola *usage* CPU tetap dapat mengikuti seperti distribusi Poisson yang telah di*generate* sebelumnya.

4.4.2 Uji Coba Sistem dengan Metode Prediksi

Yang akan menjadi tolak ukur pengujian pada tesis ini adalah pada beban naik dan pada beban turun. Nantinya akan dilihat bagaimana tingkat ketersedian dan konsumsi daya pada beban naik dan turun tersebut. Sesuai pemodelan distribusi beban yang telah dibahas sebelumnya, di mana dalam distribusi beban selama 1 jam tersebut memiliki dua pola, yaitu pola beban naik atau meningkat dan pola beban menurun.

Selanjutnya akan dilanjutkan dengan pengujian *response time* dan konsumsi daya. Uji coba akan dilakukan dengan dua skenario. Skenario pertama adalah pada pola peningkatan beban, dan skenario kedua adalah pada pola penurunan beban. Nantinya uji coba akan dibagi menjadi 2 tahap yaitu pada peningkatan beban selama 30 menit dan pada uji coba penurunan beban selama 30 menit pula.

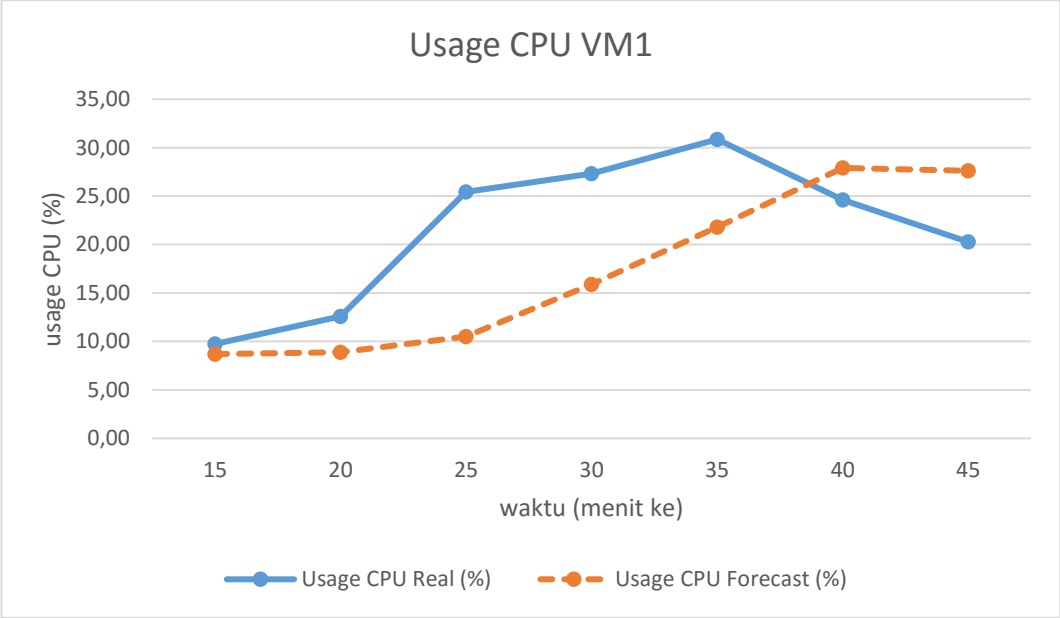
4.4.2.1 Uji Coba Akurasi Metode pada Beban Meningkat dengan Metode Prediksi

Seperti yang telah dibahas sebelumnya, uji coba dilakukan selama 30 menit. Setelah melakukan beberapa pengujian sebelumnya, maka pengujian konsumsi daya pada saat peningkatan beban dapat dilakukan dari menit ke-15 sampai dengan menit ke-45. Pemilihan dari menit ke-15 sampai ke-45 menunjukkan peningkatan beban yang disebabkan oleh trafik *request* yang meningkat, oleh karena itu durasi pada menit tersebut dipilih untuk pengujian pada pola peningkatan beban.

Tahap awal yang akan dikerjakan oleh sistem adalah mencari nilai prediksi atau *forecast* dari kedua VM (VM1 dan VM2), pada Tabel 4.4 dan Tabel 4.5 menunjukkan hasil *forecast* pada VM1.

Tabel 4.4 Hasil *Forecast* (Prediksi) Nilai *Usage* CPU VM1 pada Beban Meningkat

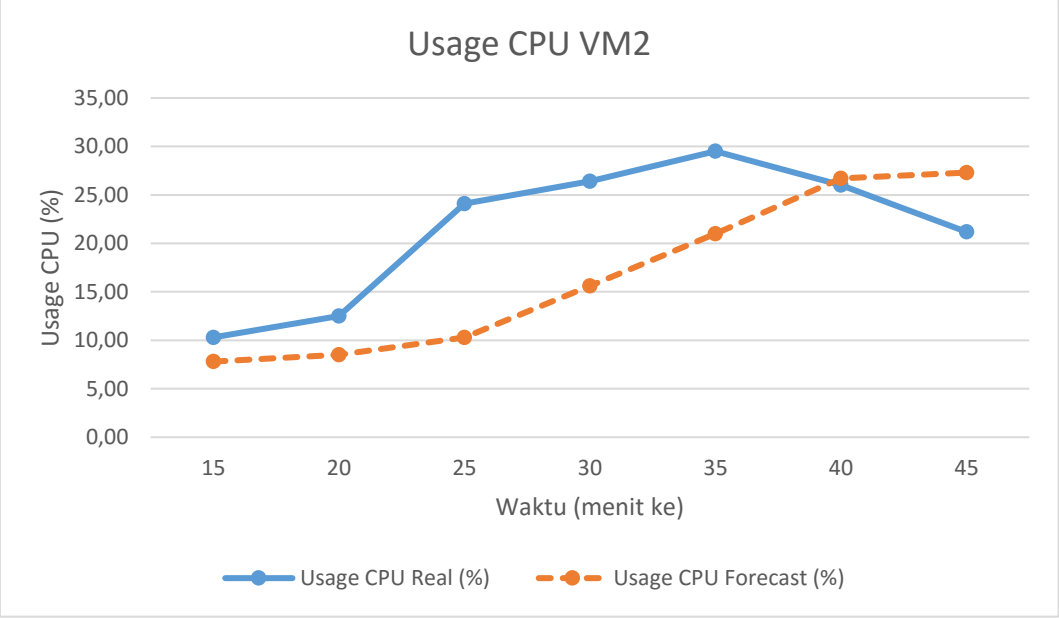
Menit	Usage CPU VM 1	
	Real (%)	Forecast (%)
15	9,74	8,7
20	12,57	8,9
25	25,42	10,5
30	27,30	15,9
35	30,85	21,8
40	24,60	27,9
45	20,29	27,6



Gambar 4.6 Grafik *Forecast Usage* CPU VM1 pada Beban Meningkat

Tabel 4.5 Hasil *Forecast* (Prediksi) Nilai *Usage* CPU VM1 pada Beban Meningkat

Menit	Usage CPU VM 1	
	Real (%)	Forecast (%)
15	10,30	7,8
20	12,50	8,5
25	24,10	10,3
30	26,40	15,6
35	29,50	21,0
40	26,00	26,7
45	21,20	27,3



Gambar 4.7 Grafik *Forecast Usage* CPU VM2 pada Beban Meningkat

Dari grafik yang ditunjukkan pada Gambar 4.6 dan Gambar 4.7 memiliki nilai yang hampir sama, ini disebabkan oleh trafik *request* yang diberikan juga sama

antara VM1 dan VM2. Dari kedua gambar grafik tersebut juga terlihat bahwa metode prdiksi yang diaplikasikan dapat mengikuti nilai *real time*, Sehingga membentuk *tren* dengan pola naik.

Tahap *forecast* atau prediksi merupakan langkah yng menentukan migrasi, DNS (*Dynamics Shutdown)*, dan *up sever*. Sebelum migrasi dan DNS, maka ditentukan nilai *threshold* yaitu 50%. Nilai 50% adalah *threshold* yang ditentukan karena ini adalah nilai yang memungkinkan untuk diterapkan pada lingkungan pengujian, sebenarnya hal ini lebih disebabkan oleh spesifikasi laptop yang digunakan selama pengujian memiliki keterbatasan. Sesuai pengujian sebelumnya jika batas tersebut terlalu besar makan akan menyebabkan laptop *crash* yang berakibat pada kegagalan uji coba. Namun batas ini tidak terlalu berpengaruh untuk melihat efektifitas pengujian sistem, karena yang akan dilihat dari pengujian adalah menemukan perbandingan saja antara sistem yang menggunakan prediksi dengan sistem konvensional.

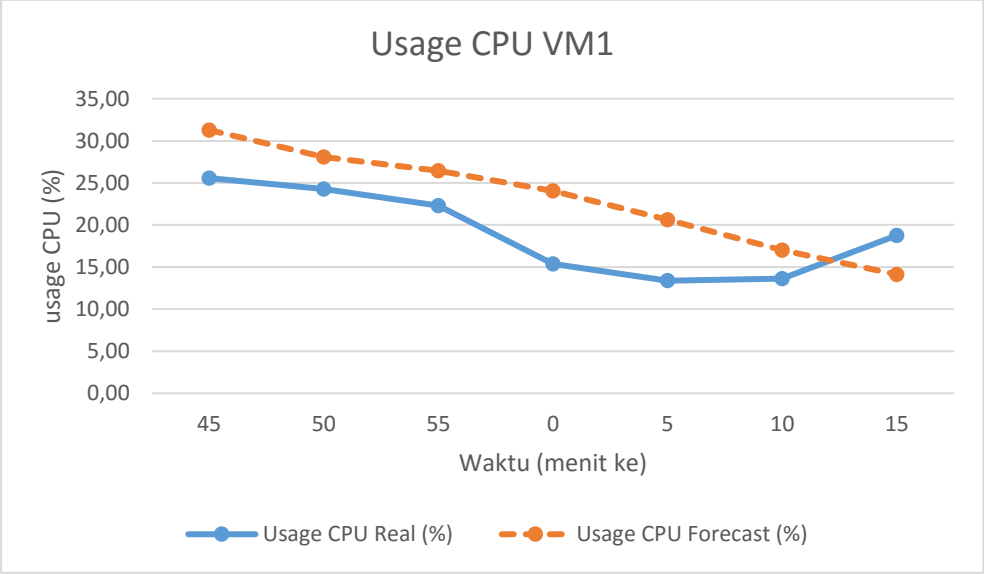
Pada beban naik akan ditentukan jadwal *up* kembali server *worker* serta migrasi server VM. Sistem akan menjadwalkan *up* server *worker* pada menit ke-30 karena pada menit tersebut total jumlah *usage* CPU kedua VM adalah melebihi 50%. Dan pada menit tersebut diprediksikan nilai akan naik, sehingga diputuskan untuk melakukan *up* server, dan selanjutnya setelah server *up*, akan dilakukan migrasi VM ke server *worker* yang baru saja dihidupkan tersebut. Proses ini akan berlanjut sampai diprediksikan nilai *usage* kembali menurun yaitu pada menit ke-45.

4.4.2.2 Uji Coba Akurasi Metode pada Beban Menurun dengan Metode Prediksi

Sama seperti pengujian sebelumnya pada peningkatan beban, uji coba pada saat penurunan beban juga dilakukan selama 30 menit, namun yang berbeda adalah durasi waktu yang diambil berada di antara menit ke-45 sampai dengan menit ke-15. Ini merupakan kelanjutan pengujian sebelumnya, di mana dalam 1 jam (60 menit) terdapat dua pola beban yang akan diuji cobakan.

Tabel 4.6 Hasil *Forecast* (Prediksi) Nilai *Usage* CPU VM1 pada Beban Menurun

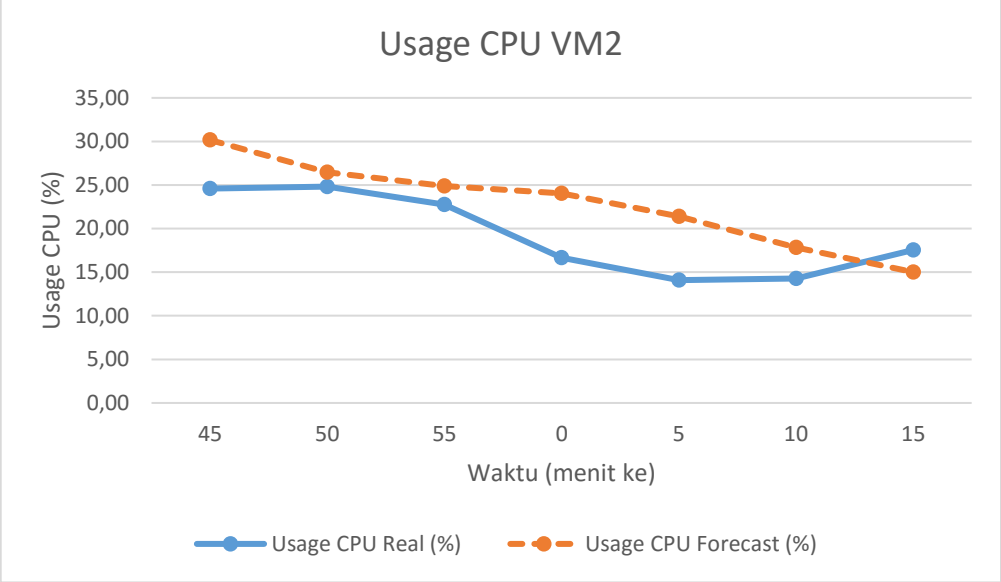
Menit	Usage CPU VM 1	
	Real (%)	Forecast (%)
45	25,57	31,29
50	24,27	28,07
55	22,30	26,45
00	15,36	24,05
05	13,38	20,64
10	13,63	17,01
15	18,77	14,12



Gambar 4.8 Grafik *Forecast Usage* CPU VM1 pada Beban Menurun

Tabel 4.7 Hasil *Forecast* (Prediksi) Nilai *Usage* CPU VM2 pada Beban Menurun

Menit	Usage CPU VM 1	
	Real (%)	Forecast (%)
45	24,58	30,16
50	24,82	26,47
55	22,76	24,88
00	16,68	24,05
05	14,09	21,42
10	14,28	17,84
15	17,55	15,02



Gambar 4.9 Grafik *Forecast Usage* CPU VM2 pada Beban Meningkat

Dari kedua nilai *forecast* pada beban menurun, terlihat bahwa *forecast* pada beban menurun memiliki hasil yang lebih baik daripada beban meningkat. Dari kedua hasil *forecast* ini, selanjutnya akan dijadwalkan migrasi VM dan *shutdown* server *worker* secara otomatis.

Jadwal migrasi pada beban rendah ini akan terjadi pada menit ke-55, di mana pada menit ini nilai prediksi adalah di bawah 50% yaitu telah melebihi nilai *threshold* yang telah dijelaskan sebelumnya. Selanjutnya akan dilakukan eksekusi DNS server agar konsumsi daya dapat diterapkan. Server akan terus dalam posisi *shutdown* selama nilai *usage* CPU yang terbaca dan yang diprediksi masih di bawah *threshold*.

4.4.2.3 Uji Coba Konsumsi Daya pada Beban Meningkat dengan Metode Prediksi

Pada uji coba konsumsi daya, digunakan *tool* Powerstat yang mensyaratkan sistem mendukung ACPI (Advanced Configuration and Power Interface), sehingga laptop harus running menggunakan baterai, Powerstat tidak dapat bekerja jika laptop sedang dalam pengisian daya.

Uji coba dilakukan selama 30 menit. Tahap awal yang akan dikerjakan oleh sistem adalah mencari nilai prediksi atau *forecast* dari kedua VM (VM1 dan VM2) seperti yang telah dibahas pada pengujian sebelumnya.

Uji coba konsumsi daya menggunakan Powerstat dapat dilakukan pada salah satu server. Artinya pengujian ini dapat dilakukan pada server *worker, controller* atau bahkan VM. Hasil yang ditunjukkan memiliki nilai yang sama. dalam hal ini menggunakan VMware sebagai lingkungan uji cobanya. Penulis menganalisa dari beberapa pengujian yang telah dilakukan bahwa Powerstat akan membaca nilai ACPI langsung dari laptop (komputer fisik) bukan dari *virtual*, sehingga pada server *virtual* yang terinstal di dalam VMware hanya menerima *report* ACPI dari komputer fisik saja. Namun penggunaan *resource* yang terpakai oleh server *virtual* tetap akan mempengaruhi konsumsi daya pada komputer fisik.

Pengujian konsumsi daya dengan cara ini harus benar-benar mempertimbangkan penggunaan *resource* yang digunakan pada setiap *service* ataupun aplikasi yang sedang berjalan pada komputer fisik, oleh karena itu sebelum uji coba dilakukan, *service* ataupun aplikasi yang tidak digunakan untuk keperluan uji coba ini harus dinonaktifkan terlebih dahulu.

Dari hasil uji coba pada Gambar 4.10 menunjukkan bahwa penggunaan daya selama uji coba pada pola beban meningkat adalah rata-rata 13,25 Watt. Durasi uji coba sama seperti uji coba beban meningkat yang telah dijelaskan sebelumnya yaitu selama 30 menit, dimulai dari menit ke-15 sampai dengan menit ke-45. Uji coba dilakukan menggunakan Powerstat pada server *controller*, sehingga nilai *usage* CPU yang ditampilkan tidak sama dengan *usage* CPU pada server *worker* ataupun VM, namun hal ini tidak berpengaruh pada nilai konsumsi daya (Watt) yang dihasilkan. Watt yang dihasilkan tetap mengikuti ACPI komputer fisik, sehingga semakin besar penggunaan *resource* pada virtual (VMware) yang diakibatkan oleh pengaruh pemakaian *resource* pada server *worker* dan VM, maka semakin besar pula konsumsi daya yang digunakan.

```
Running for 2100 seconds (7 samples at 300 second intervals).
ACPI battery power measurements will start in 0 seconds time

  Time    User  Nice   Sys  Idle    IO  Run Ctxt/s  IRQ/s Fork Exec Exit  Watts
03:15:01   0.3   0.0   1.0  86.6  12.1    4     87     73  198  109  196  12.68
03:20:01   0.3   0.0   0.9  88.3  10.6    1     96     77  206  119  205   9.50
03:25:01   0.3   0.0   0.8  87.5  11.4    1     95     77  194  108  195  13.11
03:30:01   0.3   0.0   1.1  92.3   6.2    2     96     77  279  149  279  14.61
03:35:01   0.3   0.0   1.0  90.6   8.1    2     91     76  201  114  200  12.54
03:40:01   0.3   0.0   1.0  91.5   7.2    1     89     72  212  123  213  16.72
03:45:01   0.3   0.0   1.0  89.2   9.5    1     90     74  204  116  199  13.59
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
 Average   0.3   0.0   1.0  89.4   9.3  1.7   92.2   75.4 213.4 119.7 212.4  13.25
  StdDev   0.0   0.0   0.1   2.0   2.0  1.0    3.2    2.1 27.3 12.9 27.8   2.03
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
 Minimum   0.3   0.0   0.8  86.6   6.2  1.0   87.5   72.4 194.0 108.0 195.0   9.50
 Maximum   0.3   0.0   1.1  92.3  12.1  4.0   96.4   77.4 279.0 149.0 279.0  16.72
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
Summary:
 13.25 Watts on Average with Standard Deviation 2.03
```

Gambar 4.10 Hasil Uji Coba Konsumsi Daya pada Beban Meningkat

Dari keseluruhan pengujian konsumsi daya yang dilakukan dengan menggunakan metode prediksi, maka hasil dari percobaan tersebut dapat dianalisa lebih jelas seperti Tabel 4.8 dan 4.9 di bawah ini.

Tabel 4.8 Konsumsi Daya pada Beban Meningkat

Menit	**Usage CPU (%)**				**Daya (Watt)**
	VM1	**VM2**	**worker1**	**worker2**	
15	15,87	15,67	0,00	42,33	12,68
20	26,94	25,22	0,00	43,67	9,5
25	44,16	46,20	46,83	33,76	13,11
30	31,83	31,00	42,95	27,44	14,61
35	32,85	32,62	43,15	27,93	12,54
40	30,07	29,29	42,50	28,39	16,72
45	27,42	27,14	41,63	28,75	13,59
Total	**209,14**	**207,15**	**217,06**	**232,26**	**92,75**
Rata-rata	**29,88**	**29,59**	**31,01**	**33,18**	**13,25**

Dari Tabel 4.8 terlihat jelas bahwa penghematan konsumsi daya dapat dilakukan pada sistem ini. Konsumsi daya dapat dikurangi dari menit ke-15 sampai dengan menit ke-20, nilai konsumsi daya pada menit ini adalah 12,68 Watt dan 9,5

Watt, ini jauh lebih kecil jika dibandingkan dengan nilai konsumsi pada menit-menit yang lain. Pada menit ke-15 sampai dengan menit ke-20 hanya satu server *worker* yang bekerja yaitu srver *worker* 2, kedua VM akan berada dalam server *worker* 2, sehingga penghematan konsumsi daya dapat dilakukan.

4.4.2.4 Uji Coba Konsumsi Daya pada Beban Menurun dengan Metode Prediksi

Pengujian konsumsi daya pada beban meningkat dilakukan mengikuti pengujian sebelumnya yaitu selama 30 menit dari menit ke-45 sampai dengan menit ke-15. Seperti hasil yang ditunjukkan pada Gambar 4.11 bahwa nilai rata-rata pengujian adalah 11, 57 Watt.

Pada Gambar 4.11, hasil uji coba menunjukkan bahwa nilai konsumsi daya lebih kecil daripada nilai konsumsi daya pada beban meningkat, penulis menganalisa hal ini disebabkan karena pada beban menurun, maka salah satu server akan di*shutdown,* sehingga pemakain *resource* juga akan ikut berkurang dan akan berefek pada konsumsi daya secara keseluruhan. Dan ini adalah sesuai dengan hipotesa yang dianalisa sebelumnya.

```
Running for 2100 seconds (7 samples at 300 second intervals).
ACPI battery power measurements will start in 0 seconds time

  Time    User  Nice   Sys  Idle    IO  Run Ctxt/s  IRQ/s Fork Exec Exit  Watts
04:45:01   0.2   0.0   1.0  93.6   5.3    2     85     81   57   54   54   9.75
04:50:01   0.2   0.0   0.9  95.3   3.5    1     86     81   48   48   49  11.04
04:55:01   0.2   0.0   1.2  91.0   7.6    3     85     82   48   46   46  12.45
05:00:01   0.2   0.0   1.3  93.7   4.7    1     90     86   52   52   54  11.60
05:05:01   0.3   0.0   1.6  88.6   9.5    1     87     84   41   40   40  10.30
05:10:01   0.2   0.0   1.5  87.5  10.8    1     93     88   62   59   63  12.96
05:15:01   0.2   0.0   1.1  90.3   8.4    1     93     88   50   48   45  12.89
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
 Average   0.2   0.0   1.2  91.4   7.1  1.4   88.2   84.3 51.1 49.6 50.1  11.57
  StdDev   0.0   0.0   0.2   2.7   2.5  0.7    3.3    2.8  6.3  5.7  7.0   1.17
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
 Minimum   0.2   0.0   0.9  87.5   3.5  1.0   84.7   81.3 41.0 40.0 40.0   9.75
 Maximum   0.3   0.0   1.6  95.3  10.8  3.0   92.8   87.9 62.0 59.0 63.0  12.96
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
Summary:
 11.57 Watts on Average with Standard Deviation 1.17
```

Gambar 4.11 Hasil Uji Coba Konsumsi Daya pada Beban Menurun

Tabel 4.9 Konsumsi Daya pada Beban Menurun

Menit	Usage CPU (%)				Daya (Watt)
	VM1	VM2	worker1	worker2	
45	25,57	24,58	38,48	28,47	9,75
50	24,27	24,82	38,30	28,64	11,04
55	22,30	22,76	37,86	28,65	12,45
00	15,36	16,68	47,46	50,46	11,60
05	13,38	14,09	42,46	0,00	10,30
10	13,63	14,28	44,25	0,00	12,96
15	18,77	17,55	46,10	0,00	12,89
Total	**133,27**	**134,75**	**294,91**	**136,22**	**80,99**
Rata-rata	**19,04**	**19,25**	**42,13**	**19,46**	**11,57**

Nilai konsumsi daya beban menurun yang ditunjukkan pada Tabel 4.9 memiliki nilai rata-rata lebih rendah dari pada nilai rata-rata konsumsi daya pada beban meningkat. Dan ini memang hasil yang diharapkan, di mana dengan beban menurun juga akan mengurangi penggunaan *resource* CPU sehingga dengan begitu pula penggunaa ataupun konsumsi daya dapat dikurangi.

4.4.2.5 Uji Coba *Response Time* pada Beban Meningkat

Yang menjadi layanan pada sistem *cloud computing* adalah ketersediaan pada server VM, server VM harus benar-benar dijaga dan diupayakan agar tidak menimbulkan *response time* yang terlalu tinggi. Jika *response time* terlalu tinggi, maka dapat dianggap bahwa ketersediaan dari sistem *cloud computing* tersebut tidak baik. Maka dari itu pengujian *response time* dilakukan pada server VM. Pengujian juga dilakukan dengan cara memberikan *request* http menggunakan *tool* Httperf selama pengujian. Namun *request* yang diberikan hanya 1 *request* perdetik selama 30 menit (sesuai waktu pengujian), 1 *request* perdetik ini bertujuan untuk mencegah agar tidak menimbulkan kenaikan *usage* CPU pada server VM.

Pengujian *response time* dilakukan pada dua server VM, yaitu server VM1 dan server VM2, masing-masing diberikan 1800 *request* dalam durasi 30 menit (1800 detik) dan dan *request* perdetik adalah 1 *request*. Sesuai dengan pengujian sebelumnya pada beban meningkat, maka pengujian dilakukan dari menit ke-15

sampai dengan menit ke-45. Hasil pengujian dapat dilihat sperti pada Gambar 4.12 dan Gambar 4.13.

```
httperf --client=0/1 --server=192.168.1.175 --port=80 --uri=/index.php --rate=1 --
send-buffer=4096 --recv-buffer=16384 --num-conns=1800 --num-calls=1
Maximum connect burst length: 1

Total: connections 1800 requests 1800 replies 1800 test-duration 1799.034 s

Connection rate: 1.0 conn/s (999.5 ms/conn, <=6 concurrent connections)
Connection time [ms]: min 1.5 avg 85.7 max 15063.4 median 23.5 stddev 496.6
Connection time [ms]: connect 35.3
Connection length [replies/conn]: 1.000

Request rate: 1.0 req/s (999.5 ms/req)
Request size [B]: 75.0

Reply rate [replies/s]: min 0.2 avg 1.0 max 1.6 stddev 0.1 (358 samples)
Reply time [ms]: response 50.4 transfer 0.0
Reply size [B]: header 184.0 content 286.0 footer 0.0 (total 470.0)
Reply status: 1xx=0 2xx=0 3xx=0 4xx=1800 5xx=0

CPU time [s]: user 71.94 system 249.94 (user 4.0% system 13.9% total 17.9%)
Net I/O: 0.5 KB/s (0.0*10^6 bps)
```

Gambar 4.12 *Response Time* VM1 pada Beban Meningkat

```
httperf --client=0/1 --server=192.168.1.176 --port=80 --uri=/index.php --rate=1 --
send-buffer=4096 --recv-buffer=16384 --num-conns=1800 --num-calls=1
Maximum connect burst length: 1

Total: connections 1800 requests 1800 replies 1800 test-duration 1799.030 s

Connection rate: 1.0 conn/s (999.5 ms/conn, <=11 concurrent connections)
Connection time [ms]: min 1.5 avg 117.1 max 31102.7 median 24.5 stddev 973.2
Connection time [ms]: connect 49.9
Connection length [replies/conn]: 1.000

Request rate: 1.0 req/s (999.5 ms/req)
Request size [B]: 75.0

Reply rate [replies/s]: min 0.0 avg 1.0 max 2.0 stddev 0.1 (358 samples)
Reply time [ms]: response 67.2 transfer 0.0
Reply size [B]: header 184.0 content 286.0 footer 0.0 (total 470.0)
Reply status: 1xx=0 2xx=0 3xx=0 4xx=1800 5xx=0

CPU time [s]: user 73.44 system 248.44 (user 4.1% system 13.8% total 17.9%)
Net I/O: 0.5 KB/s (0.0*10^6 bps)
```

Gambar 4.13 *Response Time* VM2 pada Beban Meningkat

Dari kedua hasil *log* yang diperlihatkan pada Gambar 4.12 dan Gambar 4.13, maka dapat diperhatikan beberapa nilai seperti yang disajikan dalam Tabel 4.10 berikut ini.

Tabel 4.10 Hasil *Response Time* Beban Meningat pada VM1 dan VM2 dengan Metode Prediksi

VM	VM1	Jumlah Koneksi	Durasi Koneksi (s)	Response Time (ms)
VM1	192.168.1.175	1800	1799,034	50,04
VM2	192.168.1.176	1800	1799,030	67,2

Pada kedua uji coba *response time* yaitu pada VM1 dan VM2 menunjukkan ada perbedaan nilai *response time* antara keduanya. Dari Gambar 4.12 dan Gambar 4.13 menunjukka hasil *response time* yang telah diuji coba pada beban meningkat. Nilai antara kedua VM memiliki sedikit perbedaan, VM1 dapat mengerjakan 1800 *request* selama 1799.034 s, sedangkan VM2 dapat mengerjakan 1800 *request* selama 1799.030 s. Namun nilai *response time* pada VM1 adalah 50,4 ms dan nilai *response time* pada VM2 adalah 67,2 ms. Semakin kecil nilai *response time*, maka semakin bagus pula ketersedian layanan yang diberikan oleh sistem.

4.4.2.6 Uji Coba *Response Time* pada Beban Menurun

Sama seperti uji coba *response time* sebelumnya, uji coba *response time* berikutnya dilakukan pada beban menurun, pengujian tetap dilakukan pada kedua server VM. Yaitu server VM1 dan server VM2, dan akan dilihat berapa nilai yang dihasilkan dari masing-masing VM tersebut. Server VM1 beralamat di 192.168.1.175, sedangkan VM2 beralamat di 192.168.1.176.

```
httperf --client=0/1 --server=192.168.1.175 --port=80 --uri=/index.php --rate=1 --
send-buffer=4096 --recv-buffer=16384 --num-conns=1800 --num-calls=1
Maximum connect burst length: 1

Total: connections 1800 requests 1800 replies 1800 test-duration 1799.007 s

Connection rate: 1.0 conn/s (999.4 ms/conn, <=3 concurrent connections)
Connection time [ms]: min 1.6 avg 35.9 max 3058.9 median 22.5 stddev 126.3
Connection time [ms]: connect 11.5
Connection length [replies/conn]: 1.000

Request rate: 1.0 req/s (999.4 ms/req)
Request size [B]: 75.0

Reply rate [replies/s]: min 0.8 avg 1.0 max 1.2 stddev 0.0 (358 samples)
Reply time [ms]: response 24.4 transfer 0.0
Reply size [B]: header 184.0 content 286.0 footer 0.0 (total 470.0)
Reply status: 1xx=0 2xx=0 3xx=0 4xx=1800 5xx=0

CPU time [s]: user 78.52 system 246.88 (user 4.4% system 13.7% total 18.1%)
Net I/O: 0.5 KB/s (0.0*10^6 bps)
```

Gambar 4.14 *Response Time* VM1 pada Beban Menurun

```
httperf --client=0/1 --server=192.168.1.176 --port=80 --uri=/index.php --rate=1 --
send-buffer=4096 --recv-buffer=16384 --num-conns=1800 --num-calls=1
Maximum connect burst length: 1

Total: connections 1800 requests 1800 replies 1800 test-duration 1799.012 s

Connection rate: 1.0 conn/s (999.5 ms/conn, <=1 concurrent connections)
Connection time [ms]: min 1.8 avg 28.0 max 712.0 median 23.5 stddev 40.4
Connection time [ms]: connect 6.0
Connection length [replies/conn]: 1.000

Request rate: 1.0 req/s (999.5 ms/req)
Request size [B]: 75.0

Reply rate [replies/s]: min 0.8 avg 1.0 max 1.2 stddev 0.0 (358 samples)
Reply time [ms]: response 21.9 transfer 0.0
Reply size [B]: header 184.0 content 286.0 footer 0.0 (total 470.0)
Reply status: 1xx=0 2xx=0 3xx=0 4xx=1800 5xx=0

CPU time [s]: user 77.05 system 248.35 (user 4.3% system 13.8% total 18.1%)
Net I/O: 0.5 KB/s (0.0*10^6 bps)
```

Gambar 4.15 *Response Time* VM2 saat Beban Menurun

Dari kedua hasil *log response time* pada beban menurun yang diperlihat pada Gambar 4.14 dan Gambar 4.15, maka dapat diperhatikan beberapa nilai seperti yang disajikan dalam Tabel 4.11 berikut ini.

Tabel 4.11 Hasil *Response Time* Beban Menurun pada VM1 dan VM2 dengan Metode Prediksi

VM	**VM1**	**Jumlah Koneksi**	**Durasi Koneksi (s)**	**Response Time (ms)**
VM1	192.168.1.175	1800	1799,007	24,4
VM2	192.168.1.176	1800	1799,012	21,9

Gambar 4.14 dan Gambar 4.15 merupakan hasil uji coba *response time* pada kedua VM pada beban menurun. dari hasil tersebut menunjukkan bahwa nilai *response time* pada beban menurun lebih kecil daripada nilai *response time* pada beban meningkat. Menurut analisa penulis, ini adalah hal yang wajar karena beban menurun, maka trafik akan semakin lancar dan pada saat tersebut prosesor akan bekerja lebih ringan dibandingkan beban meningkat.

4.4.3 Uji Coba Sistem dengan Metode Konvensional

Untuk memperoleh hasil sebagai nilai perbandingan dengan metode prediksi yang telah diuji coba pada sebelumnya, maka perlu ada pengujian selanjutnya yang dilakukan tanpa menggunakan metode. Pengujian ini mengadopsi *sistem cloud computing* pada umumnya, di mana sistem tidak akan melakukan migrasi VM, dan tidak akan melakukan *shutdown* otomatis. Pada pengujian ini, penulis menyebutnya dengan uji coba dengan metode konvensional. Pengujian tetap akan dilakukan pada dua skenario pola beban yang telah ditentukan, yaitu pada pola beban meningkat, dan pada pola beban menurun.

4.4.3.1 Uji Coba Konsumsi Daya pada Beban Meningkat dengan Metode Konvensional

Uji coba pada beban meningkat dilakukan pada menit ke-15 sampai dengan menit ke-45. Setelah uji coba konsumsi daya dilakukan, maka akan dilihat nilai rata-rata dari konsumsi daya tersebut.

```
Running for 2100 seconds (7 samples at 300 second intervals).
ACPI battery power measurements will start in 0 seconds time

  Time    User  Nice   Sys  Idle    IO  Run Ctxt/s  IRQ/s Fork Exec Exit  Watts
08:15:01   0.3   0.0   2.2  90.7   6.8    2    104    104  208  121  203  11.90
08:20:01   0.1   0.0   1.6  91.0   7.2    2     98     97  151   64  153  12.94
08:25:01   0.3   0.0   1.7  91.4   6.7    1    112    104  215  122  210  13.59
08:30:01   0.2   0.0   1.3  92.6   5.9    2     98     90  222   93  224  12.39
08:35:01   0.2   0.0   1.2  93.4   5.2    1     87     85  152   64  152  12.05
08:40:01   0.2   0.0   1.4  92.3   6.1    1    100     97  164   75  161  13.38
08:45:01   0.2   0.0   1.4  90.8   7.7    7    106    104  150   62  148  13.23
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
 Average   0.2   0.0   1.5  91.7   6.5  2.3  100.9   97.2 180.3 85.9 178.7  12.78
  StdDev   0.1   0.0   0.3   1.0   0.8  2.0    7.2    7.1 30.6 24.6 29.9   0.62
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
 Minimum   0.1   0.0   1.2  90.7   5.2  1.0   87.5   84.7 150.0 62.0 148.0  11.90
 Maximum   0.3   0.0   2.2  93.4   7.7  7.0  112.4  104.4 222.0 122.0 224.0  13.59
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
Summary:
 12.78 Watts on Average with Standard Deviation 0.62
```

Gambar 4.16 Hasil Uji Coba Konsumsi Daya ketika Trafik Meningkat

Nilai hasil konsumsi daya selama 30 menit pada beban meningkat yang ditunjukkan dari Gambar 4.16 memiliki nilai rata-rata 12,78 Watt.

Tabel 4.12 Konsumsi Daya pada Beban Meningkat

Menit	Usage CPU (%)				Daya (Watt)
	VM1	VM2	worker1	worker2	
45	9,79	9,55	26,24	25,33	11,90
50	13,37	12,74	27,70	26,43	12,94
55	19,67	18,19	30,15	30,31	13,59
00	26,53	25,18	31,39	27,49	12,39
05	30,39	30,14	29,26	29,78	12,05
10	33,90	31,22	29,65	29,08	13,38
15	33,23	30,68	30,20	29,37	13,23
Total	**166,88**	**157,70**	**204,60**	**197,79**	**89,48**
Rata-rata	**23,84**	**22,53**	**29,23**	**28,26**	**12,78**

4.4.3.2 Uji Coba Konsumsi Daya pada Beban Menurun dengan Metode Konvensional

Sama seperti pengujian sbelumnya, pengujian ini dilakukan selama 30 menit. Konsumsi daya yang digunakan pada beban menurun seperti yang ditunjukkan pada Gambar 4.17 adalah 13.18 Watt.

```
Running for 2100 seconds (7 samples at 300 second intervals).
ACPI battery power measurements will start in 0 seconds time

  Time    User  Nice   Sys  Idle    IO  Run Ctxt/s  IRQ/s Fork Exec Exit  Watts
10:45:01   0.3   0.0   1.9  94.0   3.8    2    125    118  272  146  273  17.60
10:50:01   0.2   0.0   1.7  91.2   6.9    1    111    109  157   69  153  11.75
10:55:01   0.2   0.0   1.8  90.6   7.5    1    113    113  149   62  148  11.50
11:00:01   0.2   0.0   2.0  92.0   5.9    1    113    113  151   64  152  12.42
11:05:01   0.2   0.0   2.7  94.5   2.6    1    119    116  223   93  222  10.40
11:10:01   0.2   0.0   2.5  91.0   6.3    1    117    117  162   73  163  14.31
11:15:01   0.2   0.0   2.5  90.3   7.0    2    120    120  154   66  149  14.30
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
 Average   0.2   0.0   2.1  91.9   5.7  1.3  116.7  115.1 181.1 81.9 180.0  13.18
  StdDev   0.1   0.0   0.4   1.5   1.7  0.5    4.6    3.3 44.1 27.9 45.0   2.24
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
 Minimum   0.2   0.0   1.7  90.3   2.6  1.0  110.6  109.2 149.0 62.0 148.0  10.40
 Maximum   0.3   0.0   2.7  94.5   7.5  2.0  124.6  119.7 272.0 146.0 273.0  17.60
-------- ----- ----- ----- ----- ----- ---- ------ ------ ---- ---- ---- ------
Summary:
 13.18 Watts on Average with Standard Deviation 2.24
```

Gambar 4.17 Hasil Uji Coba Konsumsi Daya ketika Trafik Menurun

Tabel 4.13 Konsumsi Daya pada Beban Menurun

Menit	Usage CPU (%)				Daya (watt)
	VM1	VM2	worker1	worker2	
15	32,21	32,67	29,15	28,50	17,60
20	28,15	27,23	29,40	28,92	11,75
25	26,76	25,46	29,59	28,97	11,50
30	20,11	16,21	29,48	28,90	12,42
35	15,87	12,49	29,26	28,43	10,40
40	15,63	13,55	29,36	28,37	14,31
45	21,74	18,67	29,48	28,58	14,30
Total	**160,47**	**146,28**	**205,72**	**200,67**	**92,28**
Rata-rata	**22,92**	**20,90**	**29,39**	**28,67**	**13,18**

Dari Tabel 4.12 dan Tabel 4.13 merupakan konsumsi daya yang digunakan pada sistem selama pengujian yang tidak menggunakan metode apapun untuk melakukan migrasi dam *shutdown* otomatis, oleh karena itu masing-masing VM akan tetap berada pada pada sestiap *worker* dari mulai awal pengujian sampai berakhirnya pengujian. Hasil yang ditunjukkan adalah nilai konsumsi daya pada beban menurun adalah lebih rendah daripada konsumsi daya beban meningkat.

4.4.3.3 Uji Coba *Response Time* pada Beban Meningkat

Response time yang dihasilkan pada pengujian pada beban meningkat yang dilakukan dengan cara konvensional dapat menghasilkan nilai *response time* sebesar 52,8 ms pada VM1 dan 62,1 ms pada VM2. 1800 *request* yang diberikan selama pengujian 30 menit dapat diterima dengan baik tanpa terjadinya *error*.

```
httperf --client=0/1 --server=192.168.1.175 --port=80 --uri=/index.php --rate=1 --
send-buffer=4096 --recv-buffer=16384 --num-conns=1800 --num-calls=1
Maximum connect burst length: 5

Total: connections 1800 requests 1800 replies 1800 test-duration 1799.047 s

Connection rate: 1.0 conn/s (999.5 ms/conn, <=6 concurrent connections)
Connection time [ms]: min 1.9 avg 79.7 max 5462.9 median 27.5 stddev 236.3
Connection time [ms]: connect 26.9
Connection length [replies/conn]: 1.000

Request rate: 1.0 req/s (999.5 ms/req)
Request size [B]: 75.0

Reply rate [replies/s]: min 0.3 avg 1.0 max 2.2 stddev 0.1 (357 samples)
Reply time [ms]: response 52.8 transfer 0.0
Reply size [B]: header 184.0 content 286.0 footer 0.0 (total 470.0)
Reply status: 1xx=0 2xx=0 3xx=0 4xx=1800 5xx=0

CPU time [s]: user 62.69 system 207.00 (user 3.5% system 11.5% total 15.0%)
Net I/O: 0.5 KB/s (0.0*10^6 bps)
```

Gambar 4.18 *Response Time* VM1 pada Beban Meningkat

Dalam Gambar 4.18 menunjukkan bahwa 1800 *request* yang diberikan dapat diterima dan diresponse dengan baik. *response time* bernilai 52,8 ms. Ini merupakan pengujian pada VM1 yang beralamat di 192.168.1.175.

Agar *log response time* beban meningkat seperti pada Gambar 4.18 dan dapat lebih dipahami, maka nilai hasil *log* tersebut disajikan dalam bentuk Tabel 4.14 berikut ini.

Tabel 4.14 Hasil *Response Time* Beban Meningkat pada VM1 dan VM2 dengan Metode Konvensional

VM	**VM1**	**Jumlah Koneksi**	**Durasi Koneksi (s)**	**Response Time (ms)**
VM1	192.168.1.175	1800	1799,047	52,8
VM2	192.168.1.176	1800	1799,191	62,1

```
httperf --client=0/1 --server=192.168.1.176 --port=80 --uri=/index.php --rate=1 --
send-buffer=4096 --recv-buffer=16384 --num-conns=1800 --num-calls=1
Maximum connect burst length: 4

Total: connections 1800 requests 1800 replies 1800 test-duration 1799.191 s

Connection rate: 1.0 conn/s (999.6 ms/conn, <=6 concurrent connections)
Connection time [ms]: min 1.5 avg 104.2 max 15064.0 median 27.5 stddev 499.3
Connection time [ms]: connect 42.1
Connection length [replies/conn]: 1.000

Request rate: 1.0 req/s (999.6 ms/req)
Request size [B]: 75.0

Reply rate [replies/s]: min 0.6 avg 1.0 max 1.8 stddev 0.1 (357 samples)
Reply time [ms]: response 62.1 transfer 0.0
Reply size [B]: header 184.0 content 286.0 footer 0.0 (total 470.0)
Reply status: 1xx=0 2xx=0 3xx=0 4xx=1800 5xx=0

CPU time [s]: user 60.28 system 209.44 (user 3.4% system 11.6% total 15.0%)
Net I/O: 0.5 KB/s (0.0*10^6 bps)
```

Gambar 4.19 *Response Time* VM2 pada Beban Meningkat

Pada Gambar 4.19 merupakan pengujian *response time* pada VM2. hasil pengujian dengan memberikan *request* sejumlah 1800 *request* dapat dikerjakan selama 1799,191 detik tanpa mengalami *error*. Hasil *response time* yang ditunjukkan adalah 62,1 ms.

4.4.3.4 Uji Coba *Response Time* pada Beban Menurun

Setelah melakukan pengujian *response time* pada beban meningkat, maka selanjutnya dilakukan pengujian pada beban menurun.

```
httperf --client=0/1 --server=192.168.1.175 --port=80 --uri=/index.php --rate=1 --
send-buffer=4096 --recv-buffer=16384 --num-conns=1800 --num-calls=1
Maximum connect burst length: 1

Total: connections 1800 requests 1800 replies 1800 test-duration 1799.039 s

Connection rate: 1.0 conn/s (999.5 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 2.0 avg 34.7 max 1270.8 median 28.5 stddev 62.6
Connection time [ms]: connect 11.1
Connection length [replies/conn]: 1.000

Request rate: 1.0 req/s (999.5 ms/req)
Request size [B]: 75.0

Reply rate [replies/s]: min 0.8 avg 1.0 max 1.2 stddev 0.0 (358 samples)
Reply time [ms]: response 23.5 transfer 0.0
Reply size [B]: header 184.0 content 286.0 footer 0.0 (total 470.0)
Reply status: 1xx=0 2xx=0 3xx=0 4xx=1800 5xx=0

CPU time [s]: user 63.94 system 210.03 (user 3.6% system 11.7% total 15.2%)
Net I/O: 0.5 KB/s (0.0*10^6 bps)
```

Gambar 4.20 *Response Time* VM1 pada Beban Menurun

Pada Gambar 4.20 menunjukkan bahwa pengujian *response time* pada VM1 menghasilkan nilai 23,5 ms. Dengan total *request* adalah 1800, dan dapat dikerjakan selama 1799,039 s.

```
httperf --client=0/1 --server=192.168.1.176 --port=80 --uri=/index.php --rate=1 --
send-buffer=4096 --recv-buffer=16384 --num-conns=1800 --num-calls=1
Maximum connect burst length: 1

Total: connections 1800 requests 1800 replies 1800 test-duration 1799.028 s

Connection rate: 1.0 conn/s (999.5 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 2.1 avg 36.5 max 1276.0 median 27.5 stddev 78.0
Connection time [ms]: connect 12.3
Connection length [replies/conn]: 1.000

Request rate: 1.0 req/s (999.5 ms/req)
Request size [B]: 75.0

Reply rate [replies/s]: min 0.8 avg 1.0 max 1.2 stddev 0.0 (358 samples)
Reply time [ms]: response 24.2 transfer 0.0
Reply size [B]: header 184.0 content 286.0 footer 0.0 (total 470.0)
Reply status: 1xx=0 2xx=0 3xx=0 4xx=1800 5xx=0

CPU time [s]: user 63.60 system 210.38 (user 3.5% system 11.7% total 15.2%)
Net I/O: 0.5 KB/s (0.0*10^6 bps)
```

Gambar 4.21 *Response Time* VM2 pada Beban Menurun

Gambar 4.21 adalah pengujian *response time* pada VM2 dengan total *request* 1800, dan menghasilkan nilai *response time* 24,2 ms. Dalam Gambar 4.20 dan Gambar 4.21 memperlihatkan hasil peungujian yang dilakukan pada beban menurun yang dimulai dari menit ke-45 sampai dengan menit ke-15. Dari pengujian tersebut menghasilkan nilai *response time* pada VM1 adalah 23,5 dan *response time* pada VM2 adalah 24.2.

Berdasarkan hasil *log* yang ditampilkan pada Gambar 4.20 dan Gambar 4.21, maka hasil *log response time* beban menurun dengan metode konvensional dapat dalam Tabel 4.15 berikut.

Tabel 4.15 Hasil *Response Time* Beban Menurun pada VM1 dan VM2 dengan Metode Konvensional

VM	VM1	Jumlah Koneksi	Durasi Koneksi (s)	Response Time (ms)
VM1	192.168.1.175	1800	1799,039	23,5
VM2	192.168.1.176	1800	1799,028	24,2

4.4.4 Pembahasan dan Perbandingan Hasil Uji Coba

Subbab ini merupakan pembahasan penting dalam penelitian ini, karena akan membahas tingkat dari keberhasilan sistem yang digunakan. Pada subbab ini akan dijelaskan bagaimana perbandingan yang terjadi antara hasil uji coba dengan metode prediksi dan hasil uji coba dengan metode konvensional. Pemaparan akan berfokus pada nilai konsumsi daya dan nilai *response time* pada masing-masing pengujian. Hasil masing-masing pengujian yang dibandingkan adalah berdasarkan dua skenario uji coba yang telah ditetapkan sebelumnya yaitu pada beban meningkat dan pada beban menurun.

Tabel 4.16 Perbandingan Hasil Uji Coba pada Beban Meningkat

Metode pengujian	Rata-rata Usage CPU (%)				Response Time (ms)			Daya (Watt)
	VM1	VM2	Worker1	Worker2	VM1	VM2	Total Response Time	
Prediksi	29,88	29,59	31,01	33,18	50,4	67,2	117,6	13,25
Konvensional	23,84	22,53	29,23	28,26	52,8	62,1	114,9	12,78

Pada Tabel 4.16 menunjukkan perbandingan hasil uji coba pada beban meningkat, hasil dibandingkan antara metode pengujian yang menggunakan sistem prediksi dan secaran konvensional (tidak menggunakan metode). Konsumsi daya yang digunakan oleh sistem dengan prediksi memiliki nilai yang lebih besar daripada sistem yang diuji cobakan dengan metode konvensional. Nilai konsumsi daya dengan metode prediksi adalah 13,25 Watt sedangkan nilai konsumsi daya pada sistem konvensional adalah 12,78 Watt. Perlu diketahui bahwa dari beberapa percobaan, menulis menganalisa bahwa yang menyebabkan konsumsi daya sedikit lebih besar pada metode prediksi adalah karena pada metode prediksi melakukan *up* (menghidupkan) server *worker* secara otomatis, sehingga sistem juga akan membutuhkan konsumsi daya tambahan. Jika dihitung selisihnya adalah hanya 0,47 Watt. Hal ini justru berbeda dengan konsumsi daya yang digunakan pada beban mengalami penurunan sperti yang tertera pada Tabel 4.17. Ketika beban mengalami penurunan, maka konsumsi daya dapat dihemat lebih besar daripada selisih penggunaan daya pada beban meningkat. Pada beban menurun, sistem dengan metode prediksi dapat menghemat daya lebih besar dari 0,47 Watt.

Tabel 4.17 Perbandingan Hasil Uji Coba pada Beban Menurun

Metode pengujian	Rata-rata Usage CPU (%)				Response time (ms)			Daya (Watt)
	VM1	VM2	worker1	worker2	VM1	VM2	Total Response Time	
Prediksi	19,02	19,25	42,13	19,46	24,4	21,9	46,3	11,57
Konvensional	22,92	20,9	29,39	28,67	23,5	24,2	47,7	13,18

Pada Tabel 4.17 menunjukkan bahwa penghematan dapat dilakukan dengan metode prediksi pada beban menurun. Ini merupakan hal yang diharapkan sesuai hipotesa sebelum melakukan pengujian. Konsumsi daya dengan metode prediksi pada beban menurun ini adalah sebesar 11,57 Watt sedangkan pada sistem konvensional adalah 13,18 Watt. Ini artinya sistem dengan metode prediksi dapat menghemat konsumsi daya sebesar 1,61 Watt. Dan jika dikurangi dengan konsumsi daya pada beban meningkat yang memiliki kenaikan sebesar 0,47 Watt, maka total konsumsi energi yang dapat dihemat dengan metode prediksi ini adalah sebesar 1,14 Watt. Memang angka ini terlihat kecil, namun ini merupakan pencapaian yang sangat bagus mengingat uji coba hanya dilakukan pada lingkungan virtual dan hanya menggunkan laptop. Yang perlu diperhatikan adalah bahwa jika keberhasilan sistem dengan menggunakan metode prediksi ini dapat diaplikasikan pada server data *center cloud computing* yang sesungguhnya. Menurut penulis angka penghematan konsumsi daya yang dapat dioptimasi pada server data *center cloud computing* yang sebenarnya akan jauh lebih besar dan signifikan mengingat pada setiap server *cloud computing* yang memiliki konsumsi daya yang jauh lebih tinggi daripada konsumsi daya pada laptop sederhana.

Hal lain yang perlu dilihat adalah mengenai total jumlah *response time* yang terjadi pada sistem dengan metode prediksi memiliki nilai yang lebih rendah daripada sistem konvensional, ini artinya bahwa layanan dengan penghematan konsumsi daya yang menggunakan metode prediksi tetap dapat memiliki ketersedian tinggi. Ini dapat dilihat dari angka yang ditunjukkan pada Tabel 4.16 dan Tabel 4.17. Hanya saja pada pengujian sistem dengan metode prediksi beban meningkat memiliki *response time* yang lebih besar daripada nilai *response time* dengan sistem konvensional, namun itu hanya selisih 2,7 ms.

Tabel 4.18 Konsumsi Daya dan *Response time* Pada Dua Metode Pengujian

Metode pengujian	**Response Time Rata-rata (ms)**	**Daya (Watt)**
Prediksi	81,95	24,82
Konvensional	81,3	25,96
Selisih/penghematan	**+0,65**	**1,14**

Dari kedua metode pengujian dengan dua pola beban yang diuji coba, maka selisih konsumsi daya yang dapat dihemat dengan mengunakan metode prediksi adalah sebesar 1,14 Watt (25,96-24,82), ini merupakan selisih antara konsumsi daya dengan metode prediksi dan metode konvensional. Nilai rata-rata *response time* dari kedua pola dengan metode prediksi tersebut adalah 81,95. Namun jika dilihat nilai *response time* keseluruhan yaitu pada pola naik dan turun, maka nilai *response time* rata-rata pada sistem yang dibangun ini mengalami sedikit kenaikan yaitu sebesar rata-rata 0,65 ms.

Lampiran I

BASH SCRIPT SERVER VM

Monitor_server_vm.sh

```
#!/bin/bash
#!/usr/bin/bash

#Declaring mysql DB connection
MASTER_DB_USER='barubaru'
MASTER_DB_PASSWD='baru'
#MASTER_DB_PORT=3160
MASTER_DB_HOST='192.168.1.213'
MASTER_DB_NAME='monitor_server'

#cek informasi pada server
Date=`date +%F`
Time=`date +%T`

if [ `date +"%M"` -le 59 ] && [ `date +"%M"` -ge 55 ]; then
Time2=55
elif [ `date +"%M"` -le 54 ] && [ `date +"%M"` -ge 50 ]; then
Time2=50
elif [ `date +"%M"` -le 49 ] && [ `date +"%M"` -ge 45 ]; then
Time2=45
elif [ `date +"%M"` -le 44 ] && [ `date +"%M"` -ge 40 ]; then
Time2=40
elif [ `date +"%M"` -le 39 ] && [ `date +"%M"` -ge 35 ]; then
Time2=35
elif [ `date +"%M"` -le 34 ] && [ `date +"%M"` -ge 30 ]; then
Time2=30
elif [ `date +"%M"` -le 29 ] && [ `date +"%M"` -ge 25 ]; then
Time2=25
elif [ `date +"%M"` -le 24 ] && [ `date +"%M"` -ge 20 ]; then
Time2=20
elif [ `date +"%M"` -le 19 ] && [ `date +"%M"` -ge 15 ]; then
Time2=15
elif [ `date +"%M"` -le 14 ] && [ `date +"%M"` -ge 10 ]; then
Time2=10
elif [ `date +"%M"` -le 09 ] && [ `date +"%M"` -ge 05 ]; then
Time2=05
elif [ `date +"%M"` -le 04 ] && [ `date +"%M"` -ge 00 ]; then
Time2=00
fi

if [ "$Time2" == 60 ]; then
n_tambah=40;
       elif [ "$Time2" != 60 ]; then
       n_tambah=0;
fi

Ip_address=`/sbin/ifconfig eth0 | grep "inet addr" | awk -F: '{print $2}'
| awk '{print $1}'`
Mac_Address=`cat /sys/class/net/eth0/address`
CPU_load=`ps aux | awk {'sum+=$3;print sum'} | tail -n 1`
Memory_total=`free -m | awk 'NR==2{printf "%s", $2}'`;
Memory_usage=`free -m | awk 'NR==2{printf "%s", $3}'`;
Hostname=`hostname --long`
```

```
let ID2=`/sbin/ifconfig eth0 | grep "inet addr" | awk -F: '{print $2}' |
awk '{print $1}' | awk -F'.' '{printf $4}' && date
+"%Y%m%d%H"`$Time2+$n_tambah;

#Perintah untuk akses ke database
mysql -u$MASTER_DB_USER -p$MASTER_DB_PASSWD  -h$MASTER_DB_HOST -
D$MASTER_DB_NAME <<EOF
#$SQL_Query
INSERT INTO server_vm
(id2,Date,Time,Hostname,IP_Address,Mac_Address,CPU_load,Memory_usage,Memo
ry_total) VALUES
('$ID2','$Date','$Time','$Hostname','$Ip_address','$Mac_Address',$CPU_loa
d,$Memory_usage,'$Memory_total');
EOF
```

Lampiran II

BASH SCRIPT SERVER WORKER

Monitor_server_proxmox.sh

```
#!/bin/bash
#!/usr/bin/bash


#Declaring mysql DB connection
MASTER_DB_USER='barubaru'
MASTER_DB_PASSWD='baru'
MASTER_DB_HOST='192.168.1.213'
MASTER_DB_NAME='monitor_server'

#cek informasi pada server
Date=`date +%F`
Time=`date +%T`
Ip_address=`/sbin/ifconfig vmbr0 | grep "inet addr" | awk -F: '{print
$2}' | awk '{print $1}'`
CPU_load=`ps aux | awk {'sum+=$3;print sum'} | tail -n 1`
Memory_total=`free -m | awk 'NR==2{printf "%s", $2}'`;
Memory_usage=`free -m | awk 'NR==2{printf "%s", $3}'`;
Hostname=`hostname --long`

baca_vm=`ls /etc/pve/qemu-server | awk -F. '{print $1}'`
array=($baca_vm);
if [ "$baca_vm" != "" ]; then

       for i in "${!array[@]}"
       do
              VmNama=`cat /etc/pve/qemu-server/${array[i]}.conf | grep
"name" | awk -F: '{print $2}' | awk '{print $1}'`
              VmMac=`cat /etc/pve/qemu-server/${array[i]}.conf | grep
"net0" | awk -F= '{print $2}' | awk -F, '{print $1}'`
              VmCore=`cat /etc/pve/qemu-server/${array[i]}.conf | grep
"core" | awk -F: '{print $2}' | awk '{print $1}'`

cek_vm=(`/opt/lampp/bin/mysql -u$MASTER_DB_USER -p$MASTER_DB_PASSWD  -
h$MASTER_DB_HOST -D$MASTER_DB_NAME <<EOF
       SELECT Vm_no,Vm_nama,Vm_mac,Pve,Vm_core FROM info_vm WHERE
Vm_no='${array[i]}';
EOF`)

 Vnama=${cek_vm[6]}
 Vmac=${cek_vm[7]}
 Vpve=${cek_vm[8]}
 Vcore=${cek_vm[9]}

 Vnama1="$VmNama"
 Vmac1="$VmMac"
 Vpve1="$Hostname"
 Vcore1="$VmCore"

if [ "$cek_vm" == "" ]; then #cek apakah data dengan vm telah ada atau
tidak, jika tidak maka akan di input
/opt/lampp/bin/mysql -u$MASTER_DB_USER -p$MASTER_DB_PASSWD  -
h$MASTER_DB_HOST -D$MASTER_DB_NAME <<EOF
```

```
INSERT INTO info_vm (Vm_no,Vm_nama,Vm_mac,Pve,Vm_core) VALUES
('${array[i]}','$VmNama','$VmMac','$Hostname','$VmCore');
EOF

elif [ "$Vnama" != "$Vnama1" ] || [ "$Vmac" != "$Vmac1" ] || [ "$Vpve" !=
"Vpve1" ] || [ "$Vcore" != "Vcore1" ];
then #cek apabila database berbeda dengan data di pve maka update
/opt/lampp/bin/mysql -u$MASTER_DB_USER -p$MASTER_DB_PASSWD  -
h$MASTER_DB_HOST -D$MASTER_DB_NAME <<EOF
UPDATE info_vm SET
Vm_no='${array[i]}',Vm_nama='$VmNama',Vm_mac='$VmMac',Pve='$Hostname',Vm_
core='$VmCore' WHERE Vm_no='${array[i]}';
EOF
fi
       done

  else
  echo "salah";
fi

Vm=${array[@]}

#Perintah untuk akses ke database
/opt/lampp/bin/mysql -u$MASTER_DB_USER -p$MASTER_DB_PASSWD  -
h$MASTER_DB_HOST -D$MASTER_DB_NAME <<EOF
INSERT INTO server
(Date,Time,Hostname,IP_Address,CPU_load,Memory_usage,Memory_total,Vm)
VALUES
('$Date','$Time','$Hostname','$Ip_address',$CPU_load,$Memory_usage,'$Memo
ry_total','$Vm');
EOF
```

Lampiran III

SCRIPT PEMROGRAMAN SERVER CONTROLLER

Forecast_vm.php

```
<?php
echo '<head>
<meta http-equiv="refresh" content="10">
</head>';
include "koneksi.php";
$tabel='server_vm';

$i=0;
$select= mysql_query("SELECT * FROM `$tabel` group by id2");
while($result=mysql_fetch_array($select))
{

	$i++;
	$sql_data= mysql_query("SELECT * FROM `$tabel` where
id2=$result[id2]");
	$resultnya = mysql_num_rows($sql_data); //menghitung jumlah data
dengan id2(waktu). agar dapat menghitung nilai rata2 (harus ada 5 data)
	$resultnya2 = mysql_fetch_array($sql_data);

	if ($resultnya > 4)
	{
		$selectAkhir= mysql_query("SELECT AVG(CPU_load)  FROM
`$tabel` where id2=$result[id2]");
		$resultAkhir=round(mysql_result($selectAkhir,0),3); //fungsi
round adalah untuk membulatkan nilai. 0),3); hanya 3 angka dibelakang
koma

		if ($result['IP_Address']=='192.168.1.175')
		{
			$cpu_load1[] = $resultAkhir;
			$nilai_id1[] = $result['id2'];
			$mac1 = $result['Mac_Address'];
		}
		elseif ($result['IP_Address']=='192.168.1.176')
		{
			$cpu_load2[] = $resultAkhir;
			$nilai_id2[] = $result['id2'];
			$mac2 = $result['Mac_Address'];
		}
		elseif ($result['IP_Address']=='192.168.1.177')
		{
			$cpu_load3[] = $resultAkhir;
			$nilai_id3[] = $result['id2'];
			$mac3 = $result['Mac_Address'];
		}
	}
}
	echo '<table width = 500 border =1>
	<tr>
	<th>Nilai ke</th>
	<th>Server fisik (Host 192.168.1.175)</th>
	<th>Nilai Real</th>
	<th>Nilai Forecast</th>
	</tr>
```

```
      <tr>';

$mac=$mac1;
$cpu_load=$cpu_load1;
$nilai_id=$nilai_id1;
$arrs = implode('</br>', $cpu_load);
//echo $arrs;
$orde = 3;
$batas_forcast = count ($cpu_load)+1;
for ($j = $orde; $j <$batas_forcast; $j++)
{
      if ((empty ($cpu_load[$j])) && (empty($nilai_id[$j])))
            {
                  $cpu_load[$j] = 0;
                  $nilai_id[$j] = $nilai_id[$j-1]+5;
            }

      $nilaike[$j]=round((($cpu_load[$j-1]+$cpu_load[$j-2]+$cpu_load[$j-
3])/3),3);
      echo "
       <td>".($j+1)."</td>
       <td>".$nilai_id[$j]."</td>
       <td>".$cpu_load[$j]."</td>
       <td>".$nilaike[$j]."</td>
       </tr>";

      $query=mysql_query("SELECT * FROM forecast where
F_id2='$nilai_id[$j]'");
      $hasil=mysql_fetch_array($query);

            if (($hasil['F_id2'] != $nilai_id[$j]) ||
(empty($hasil['F_id2'])))
            {
                  mysql_query("INSERT INTO
forecast(F_id2,Mac_Address,F_real,F_forecast) VALUES
('$nilai_id[$j]','$mac',$cpu_load[$j],$nilaike[$j])");
            }
            elseif (($hasil['F_id2'] == $nilai_id[$j]) &&
($hasil['F_real'] != $cpu_load[$j]) || ($hasil['F_forecast'] !=
$nilaike[$j]))
            {
                        mysql_query("UPDATE forecast SET
F_real=$cpu_load[$j], F_forecast=$nilaike[$j] WHERE
F_id2='$nilai_id[$j]'");
            }

}echo "</br>";
      echo '<table width = 500 border =1>
      <tr>
      <th>Nilai ke</th>
      <th>Server fisik (Host 192.168.1.176)</th>
      <th>Nilai Real</th>
      <th>Nilai Forecast</th>
      </tr>
      <tr>';
$mac=$mac2;
$cpu_load=$cpu_load2;
$nilai_id=$nilai_id2;
$arrs = implode('</br>', $cpu_load);
$orde = 3;
$batas_forcast = count ($cpu_load)+1;
for ($j = $orde; $j <$batas_forcast; $j++)
{
      if ((empty ($cpu_load[$j])) && (empty($nilai_id[$j])))
```

```
            {
                  $cpu_load[$j] = 0;
                  $nilai_id[$j] = $nilai_id[$j-1]+5;
            }
      $nilaike[$j]=round(((($cpu_load[$j-1]+$cpu_load[$j-2]+$cpu_load[$j-
3])/3),3);
      echo "
       <td>".($j+1)."</td>
       <td>".$nilai_id[$j]."</td>
       <td>".$cpu_load[$j]."</td>
       <td>".$nilaike[$j]."</td>
       </tr>";

      $query=mysql_query("SELECT * FROM forecast where
F_id2='$nilai_id[$j]'");
      $hasil=mysql_fetch_array($query);

            if (($hasil['F_id2'] != $nilai_id[$j]) ||
(empty($hasil['F_id2'])))
            {
                  mysql_query("INSERT INTO
forecast(F_id2,Mac_Address,F_real,F_forecast) VALUES
('$nilai_id[$j]','$mac',$cpu_load[$j],$nilaike[$j])");
            }
            elseif (($hasil['F_id2'] == $nilai_id[$j]) &&
($hasil['F_real'] != $cpu_load[$j]) || ($hasil['F_forecast'] !=
$nilaike[$j]))
            {
                        mysql_query("UPDATE forecast SET
F_real=$cpu_load[$j], F_forecast=$nilaike[$j] WHERE
F_id2='$nilai_id[$j]'");
            }

}echo "</br>";
?>
```

Akumulasi_data.php

```
<?php
echo '<head>
<meta http-equiv="refresh" content="10">
</head>';

include "koneksi.php";
$sql = mysql_query ("SELECT DISTINCT Hostname  FROM server");
$Jumlah_row = mysql_num_rows($sql);
$i=0;
echo '<b>Akumulasi data</b>
<table width = 500 border =1>
<tr>
<th>No</th>
<th>PVE</th>
<th>Vm</th>
<th>Jmlh Vm</th>
<th>Waktu Real</th>
<th>Total CPU Usage(Real)</th>
<th>Waktu Forecast</th>
<th>Total CPU Usage (Forecast)</th>
</tr>
<tr>';

$qr1=mysql_query("SELECT * FROM info_vm where Pve='pve1.domain1.tld1'");
$vm_row1 = mysql_num_rows($qr1);
while ($rst1=mysql_fetch_array($qr1))
{
	$sql_vm1=mysql_query("SELECT * FROM forecast where
Mac_Address='$rst1[Vm_mac]' ORDER BY id DESC LIMIT 1");
	while ($rs_vm1=mysql_fetch_array($sql_vm1))
	{
		$rsvm1F[]=$rs_vm1['F_forecast'];
		$waktu1F[]=$rs_vm1['F_id2'];
	}

	if($rst1['Vm_mac']=='36:66:34:32:32:38')
	{
		$sql_vm_real1_1=mysql_query("SELECT * FROM forecast where
Mac_Address='36:66:34:32:32:38' ORDER BY id DESC LIMIT 2");
		while ($rs_vm_real1_1=mysql_fetch_array($sql_vm_real1_1))
		{
			$usage_Real1_1[]= $rs_vm_real1_1['F_real'];
		}
	}
			elseif($rst1['Vm_mac']=='62:35:34:36:30:31')
			{
				$sql_vm_real1_2=mysql_query("SELECT * FROM
forecast where Mac_Address='62:35:34:36:30:31' ORDER BY id DESC LIMIT
2");
				while
($rs_vm_real1_2=mysql_fetch_array($sql_vm_real1_2))
				{
					$usage_Real1_2[]=
$rs_vm_real1_2['F_real'];
```

```
                        }
                    }

      elseif($rst1['Vm_mac']=='66:34:33:64:31:31')
                                {

      $sql_vm_real1_3=mysql_query("SELECT * FROM forecast where
Mac_Address='66:34:33:64:31:31' ORDER BY id DESC LIMIT 2");
                                        while
($rs_vm_real1_3=mysql_fetch_array($sql_vm_real1_3))
                                        {
                                               $usage_Real1_3[]=
$rs_vm_real1_3['F_real'];
                                        }
                                }

$rst1vm_no[]=$rst1['Vm_no'];
}
if(!isset ($rsvm1F[0])) $rsvm1F[0]=0;
      if(!isset ($rsvm1F[1])) $rsvm1F[1]=0;
            if(!isset ($rsvm1F[2])) $rsvm1F[2]=0;

if(!isset ($rst1vm_no[0])) $rst1vm_no[0]=null;
      if(!isset ($rst1vm_no[1])) $rst1vm_no[1]=null;
            if(!isset ($rst1vm_no[2])) $rst1vm_no[2]=null;

if(!isset ($waktu1F[0])) $waktu1F[0]=0;
      if(!isset ($waktu1F[1])) $waktu1F[1]=0;

if(!isset ($usage_Real1_1[1])) $usage_Real1_1[1]=0;
      if(!isset ($usage_Real1_2[1])) $usage_Real1_2[1]=0;
            if(!isset ($usage_Real1_3[1])) $usage_Real1_3[1]=0;

$jmlUsageF1=$rsvm1F[0]+$rsvm1F[1]+$rsvm1F[2];
$jmlUsageR1=$usage_Real1_1[1]+$usage_Real1_2[1]+$usage_Real1_3[1];
$qr2=mysql_query("SELECT * FROM info_vm where Pve='pve2.domain2.tld2'");
$vm_row2 = mysql_num_rows($qr2);
while ($rst2=mysql_fetch_array($qr2))
{
      $sql_vm2=mysql_query("SELECT * FROM forecast where
Mac_Address='$rst2[Vm_mac]' ORDER BY id DESC LIMIT 1");
      while ($rs_vm2=mysql_fetch_array($sql_vm2))
      {
            $rsvm2F[]=$rs_vm2['F_forecast'];
            $waktu2F[]=$rs_vm2['F_id2'];
      }

      if($rst2['Vm_mac']=='36:66:34:32:32:38')
            {
                  $sql_vm_real2_1=mysql_query("SELECT * FROM forecast
where Mac_Address='36:66:34:32:32:38' ORDER BY id DESC LIMIT 2");
                  while
($rs_vm_real2_1=mysql_fetch_array($sql_vm_real2_1))
                  {
                        $usage_Real2_1[]= $rs_vm_real2_1['F_real'];
                  }
```

```
            }
                        elseif($rst2['Vm_mac']=='62:35:34:36:30:31')
                        {
                              $sql_vm_real2_2=mysql_query("SELECT *
FROM forecast where Mac_Address='62:35:34:36:30:31' ORDER BY id DESC
LIMIT 2");
                        while
($rs_vm_real2_2=mysql_fetch_array($sql_vm_real2_2))
                        {
                              $usage_Real2_2[]=
$rs_vm_real2_2['F_real'];
                        }
                        }

      elseif($rst2['Vm_mac']=='66:34:33:64:31:31')
                              {

      $sql_vm_real2_3=mysql_query("SELECT * FROM forecast where
Mac_Address='66:34:33:64:31:31' ORDER BY id DESC LIMIT 2");
                                    while
($rs_vm_real2_3=mysql_fetch_array($sql_vm_real2_3))
                                    {
                                          $usage_Real2_3[]=
$rs_vm_real2_3['F_real'];
                                    }
                              }

$rst2vm_no[]=$rst2['Vm_no'];
}
if(!isset ($rsvm2F[0])) $rsvm2F[0]=0;
      if(!isset ($rsvm2F[1])) $rsvm2F[1]=0;
            if(!isset ($rsvm2F[2])) $rsvm2F[2]=0;

if(!isset ($rst2vm_no[0])) $rst2vm_no[0]=null;
      if(!isset ($rst2vm_no[1])) $rst2vm_no[1]=null;
            if(!isset ($rst2vm_no[2])) $rst2vm_no[2]=null;

if(!isset ($waktu2F[0])) $waktu2F[0]=0;
      if(!isset ($waktu2F[1])) $waktu2F[1]=0;

if(!isset ($usage_Real2_1[1])) $usage_Real2_1[1]=0;
      if(!isset ($usage_Real2_2[1])) $usage_Real2_2[1]=0;
            if(!isset ($usage_Real2_3[1])) $usage_Real2_3[1]=0;

$jmlUsageF2=$rsvm2F[0]+$rsvm2F[1]+$rsvm2F[2];
$jmlUsageR2=$usage_Real2_1[1]+$usage_Real2_2[1]+$usage_Real2_3[1];
$qr3=mysql_query("SELECT * FROM info_vm where Pve='pve3.domain3.tld3'");
$vm_row3 = mysql_num_rows($qr3);
while ($rst3=mysql_fetch_array($qr3))
{
      $sql_vm3=mysql_query("SELECT * FROM forecast where
Mac_Address='$rst3[Vm_mac]' ORDER BY id DESC LIMIT 1");
      while ($rs_vm3=mysql_fetch_array($sql_vm3))
      {
            $rsvm3F[]=$rs_vm3['F_forecast'];
            $waktu3F[]=$rs_vm3['F_id2'];
```

```
	}

	if($rst3['Vm_mac']=='36:66:34:32:32:38')
		{
			$sql_vm_real3_1=mysql_query("SELECT * FROM forecast
where Mac_Address='36:66:34:32:32:38' ORDER BY id DESC LIMIT 2");
			while
($rs_vm_real3_1=mysql_fetch_array($sql_vm_real3_1))
			{
				$usage_Real3_1[]= $rs_vm_real3_1['F_real'];
			}
		}
				elseif($rst3['Vm_mac']=='62:35:34:36:30:31')
				{
					$sql_vm_real3_2=mysql_query("SELECT *
FROM forecast where Mac_Address='62:35:34:36:30:31' ORDER BY id DESC
LIMIT 2");
					while
($rs_vm_real3_2=mysql_fetch_array($sql_vm_real3_2))
					{
						$usage_Real3_2[]=
$rs_vm_real3_2['F_real'];
					}
				}

	elseif($rst3['Vm_mac']=='66:34:33:64:31:31')
						{

	$sql_vm_real3_3=mysql_query("SELECT * FROM forecast where
Mac_Address='66:34:33:64:31:31' ORDER BY id DESC LIMIT 2");
							while
($rs_vm_real3_3=mysql_fetch_array($sql_vm_real3_3))
							{
								$usage_Real3_3[]=
$rs_vm_real3_3['F_real'];
							}
						}
$rst3vm_no[]=$rst3['Vm_no'];
}
if(!isset ($rsvm3F[0])) $rsvm3F[0]=0;
	if(!isset ($rsvm3F[1])) $rsvm3F[1]=0;
		if(!isset ($rsvm3F[2])) $rsvm3F[2]=0;

if(!isset ($rst3vm_no[0])) $rst3vm_no[0]=null;
	if(!isset ($rst3vm_no[1])) $rst3vm_no[1]=null;
		if(!isset ($rst3vm_no[2])) $rst3vm_no[2]=null;

if(!isset ($waktu3F[0])) $waktu3F[0]=0;
	if(!isset ($waktu3F[1])) $waktu3F[1]=0;

if(!isset ($usage_Real3_1[1])) $usage_Real3_1[1]=0;
	if(!isset ($usage_Real3_2[1])) $usage_Real3_2[1]=0;
		if(!isset ($usage_Real3_3[1])) $usage_Real3_3[1]=0;

$jmlUsageF3=$rsvm3F[0]+$rsvm3F[1]+$rsvm3F[2];
$jmlUsageR3=$usage_Real3_1[1]+$usage_Real3_2[1]+$usage_Real3_3[1];
```

```
if (($waktu1F[0]-5)>100) $waktuReal_1=($waktu1F[0]-5); else
$waktuReal_1=0;
if (($waktu2F[0]-5)>100) $waktuReal_2=($waktu2F[0]-5); else
$waktuReal_2=0;
if (($waktu3F[0]-5)>100) $waktuReal_3=($waktu3F[0]-5); else
$waktuReal_3=0;

$list_vm1=$rst1vm_no[0]." ".$rst1vm_no[1]." ".$rst1vm_no[2];
$list_vm2=$rst2vm_no[0]." ".$rst2vm_no[1]." ".$rst2vm_no[2];
$list_vm3=$rst3vm_no[0]." ".$rst3vm_no[1]." ".$rst3vm_no[2];

       echo "
        <td>1</td>
        <td>Pve 1 (192.168.1.131)</td>
        <td>".$list_vm1."</td>
        <td>".$vm_row1."</td>
        <td>".$waktuReal_1."</td>
        <td>".$jmlUsageR1." %</td>
        <td>".$waktu1F[0]."</td>
        <td>".$jmlUsageF1." %</td>
        </tr>


        <td>2</td>
        <td>Pve 2 (192.168.1.132)</td>
        <td>".$list_vm2."</td>
        <td>".$vm_row2."</td>
        <td>".$waktuReal_2."</td>
        <td>".$jmlUsageR2." %</td>
        <td>".$waktu2F[0]."</td>
        <td>".$jmlUsageF2." %</td>
        </tr>

        <td>3</td>
        <td>Pve 3 (192.168.1.133)</td>
        <td>".$list_vm3."</td>
        <td>".$vm_row3."</td>
        <td>".$waktuReal_3."</td>
        <td>".$jmlUsageR3." %</td>
        <td>".$waktu3F[0]."</td>
        <td>".$jmlUsageF3." %</td>
        </tr>";
       echo '</table>';

$query=mysql_query("SELECT * FROM Akumulasi_Forecast_Real");
while($hasil=mysql_fetch_array($query))
{
       $array_waktu[]=$hasil['Waktu_Forecast'];
       $array_vm[]=$hasil['Waktu_Forecast'];
}

if (empty ($array_waktu[0]) && empty ($array_waktu[1]) && empty
($array_waktu[2]))
       {
```

```
      mysql_query("INSERT INTO
Akumulasi_Forecast_Real(Host,Vm,Jumlah_Vm,Waktu_Forecast,Total_CPU_Foreca
st,Waktu_Real,Total_CPU_Real)
                        VALUES ('Pve
1','".$list_vm1."','$vm_row1','$waktu1F[0]','$jmlUsageF1','$waktuReal_1',
'$jmlUsageR1')");

      mysql_query("INSERT INTO
Akumulasi_Forecast_Real(Host,Vm,Jumlah_Vm,Waktu_Forecast,Total_CPU_Foreca
st,Waktu_Real,Total_CPU_Real)
                        VALUES ('Pve
2','".$list_vm2."','$vm_row2','$waktu2F[0]','$jmlUsageF2','$waktuReal_2',
'$jmlUsageR2')");

      mysql_query("INSERT INTO
Akumulasi_Forecast_Real(Host,Vm,Jumlah_Vm,Waktu_Forecast,Total_CPU_Foreca
st,Waktu_Real,Total_CPU_Real)
                        VALUES ('Pve
3','".$list_vm3."','$vm_row3','$waktu3F[0]','$jmlUsageF3','$waktuReal_3',
'$jmlUsageR3')");
      }

      if (($array_waktu[0]!=$waktu1F[0]) || ($array_vm[0]!=$list_vm1))
      {
            mysql_query("UPDATE Akumulasi_Forecast_Real SET
Vm='".$list_vm1."',

      Jumlah_Vm=$vm_row1,Waktu_Forecast=$waktu1F[0],Total_CPU_Forecast=$
jmlUsageF1,Waktu_Real=$waktuReal_1,Total_CPU_Real=$jmlUsageR1 WHERE
Host='Pve 1'");
      }
            if (($array_waktu[1]!=$waktu2F[0]) ||
($array_vm[1]!=$list_vm2))
            {
                  mysql_query("UPDATE Akumulasi_Forecast_Real SET
Vm='".$list_vm2."',

      Jumlah_Vm=$vm_row2,Waktu_Forecast=$waktu2F[0],Total_CPU_Forecast=$
jmlUsageF2,Waktu_Real=$waktuReal_2,Total_CPU_Real=$jmlUsageR2 WHERE
Host='Pve 2'");
            }
                  if (($array_waktu[2]!=$waktu3F[0]) ||
($array_vm[2]!=$list_vm3))
                  {
                        mysql_query("UPDATE Akumulasi_Forecast_Real
SET Vm='".$list_vm3."',

      Jumlah_Vm=$vm_row3,Waktu_Forecast=$waktu3F[0],Total_CPU_Forecast=$
jmlUsageF3,Waktu_Real=$waktuReal_3,Total_CPU_Real=$jmlUsageR3 WHERE
Host='Pve 3'");
                  }
?>
```

Generate_schedule.php

```
<?php
echo '<head>
<meta http-equiv="refresh" content="10">
</head>';
include "koneksi.php";
date_default_timezone_set("Asia/Jakarta");
$waktu_sekarang=date("Y-m-d H:i:00");

$treshold=50; //Batas treshold
echo "</br>";
echo "</br>";

echo '<b>Generate schedule</b> </br>
<table width = 500 border =1>
<tr>
<th>No</th>
<th>Server fisik (Host)</th>
<th>Vm</th>
<th>Jmlh Vm</th>
<th>Waktu_Real</th>
<th>Total CPU Usage(Real)</th>
<th>Waktu_Forecast</th>
<th>Total CPU Usage (Forecast)</th>
</tr>
<tr>';

//Mengurutkan berdasarkan jumlah Vm
echo "- Mengurutkan berdasarkan total usage </br>";
$sql = mysql_query ("SELECT DISTINCT
Total_CPU_Forecast,Host,Vm,Jumlah_Vm,Waktu_Forecast,Waktu_Real,Total_CPU_
Real FROM Akumulasi_Forecast_Real ORDER BY Jumlah_Vm ASC");
while($row2=mysql_fetch_array($sql))
{
if ($row2['Host']!='Pve 3')
       {
                      $waktuR=$row2['Waktu_Real'];
                      $tglR = substr($waktuR,3,4)."-
".substr($waktuR,7,2)."-".substr($waktuR,9,2);
                      $jamR =
substr($waktuR,11,2).":".substr($waktuR,13,2);
                      $waktunyaR = $tglR." ".$jamR.":00";
                      $waktuF=$row2['Waktu_Forecast'];
                      if (substr($waktuF,13,2)==60)
                      {
                             $waktuF=$waktuF+40;
                      }
                      $tglF = substr($waktuF,3,4)."-
".substr($waktuF,7,2)."-".substr($waktuF,9,2);
                      $jamF =
substr($waktuF,11,2).":".substr($waktuF,13,2);
                      $waktunyaF = $tglF." ".$jamF.":00";

               $host[] = $row2['Host'];
               $vm[] = $row2['Vm'];
```

```
                $jml_vm[] = $row2['Jumlah_Vm'];
                $waktuReal[] = $waktunyaR;
                $tot_CPU_R[] = $row2['Total_CPU_Real'];
                $waktuForecast[] = $waktunyaF;
                $tot_CPU_F[] = $row2['Total_CPU_Forecast'];
        }
}

for ($i=0; $i<2; $i++)
{
        echo "
         <td>$i</td>
         <td>".$host[$i]."</td>
         <td>".$vm[$i]."</td>
         <td>".$jml_vm[$i]."</td>
         <td>".$waktuReal[$i]."</td>
         <td>".$tot_CPU_R[$i]." %</td>
         <td>".$waktuForecast[$i]."</td>
         <td>".$tot_CPU_F[$i]." %</td>
         </tr>";
}

if ($jml_vm[0]<=1 && $tot_CPU_F[0]< 1)
{
        echo "- Server ".$host[0]." tidak digunakan </br>";
}

$query = mysql_query ("SELECT * FROM schedule");
$hasil=mysql_fetch_array($query);
echo "- Status saat ini ".$hasil['S_Status']." pada
$hasil[S_Jadwal]</br>";

//mengambil nilai vm1 urutan ke-dua dari akhir
$sql_vm1=mysql_query("SELECT * FROM forecast where
Mac_Address='36:66:34:32:32:38' ORDER BY id DESC LIMIT 2");
        while ($rs_vm1=mysql_fetch_array($sql_vm1))
        {$F1[]=$rs_vm1['F_forecast'];}

//mengambil nilai vm2 urutan ke-dua dari akhir
$sql_vm1=mysql_query("SELECT * FROM forecast where
Mac_Address='62:35:34:36:30:31' ORDER BY id DESC LIMIT 2");
        while ($rs_vm1=mysql_fetch_array($sql_vm1))
        {$F2[]=$rs_vm1['F_forecast'];}

$totR_akhir= $tot_CPU_R[1]+$tot_CPU_R[0]; //Total nilai real akhir
$totF1F2= $F1[1]+$F2[1]; //Total nilai forecast ke dua terakhir
$totF_akhir= $tot_CPU_F[1]+$tot_CPU_F[0]; //Total nilai forecast terakhir

echo $waktuReal[1];
//Migrasi vm
////////////////////////////////////////////////////////////////
if (($totR_akhir < $treshold) && ($totF_akhir < $treshold))
{
        echo "- CPU usage idle ".$host[1]." = ". (100-$tot_CPU_F[1])."
%</br>";
```

```
                    if ($hasil['S_Status']!='pending_migrate'  &&
$hasil['S_Status']!='pending_migrate2')
                    {
                           if ($hasil['S_Status']!='sukses_migrate')
                           {
                                  if
(($hasil['S_Status']!='pending_shutdown') &&
($hasil['S_Status']!='pending_up'))
                                  {
                                         if
($hasil['S_Status']=='selesai')
                                         {
                                                mysql_query("UPDATE
schedule SET
S_Jadwal='$waktuReal[1]',S_Host_Awal='$host[0]',S_Action='Migrate_vm',S_V
m='$vm[0]',

       S_Host_Pindah='$host[1]',S_Status='pending_migrate' WHERE
S_Status='selesai'");
                                         }
                                         //elseif
($hasil['S_Status']!='selesai')
                                         elseif
(!isset($hasil['S_Status']) && $waktuReal[0]>0)
                                         {
                                                echo
$hasil['S_Status']."</br>";
                                                mysql_query("INSERT INTO
schedule(S_Jadwal,S_Host_Awal,S_Action,S_Vm,S_Host_Pindah,S_Status)
                                                                VALUES
('$waktuReal[1]','$host[0]','Migrate_vm','$vm[0]','$host[1]','pending_mig
rate')");
                                         }

                                  }
                           }
                    }
}


//Report sukses migrasi
//////////////////////////////////////////////////////////////////
if ($hasil['S_Status']=='sukses_migrate')
{
       if ($jml_vm[0]==0)
       {
              echo "- Shutdown Server ".$host[0];
              mysql_query("UPDATE schedule SET S_Status='pending_shutdown'
WHERE S_Status='sukses_migrate'");
       }
}
//Akhir Report sukses migrasi
//////////////////////////////////////////////////////////////////
```

```
//menghidupkan kembali server
//////////////////////////////////////////////////////////////////
$date = date_create($waktuForecast[1]);
date_add($date, date_interval_create_from_date_string('-1 minutes'));
$jam_up=date_format($date, 'Y-m-d H:i:s');
//echo $jam_up;

if (($totR_akhir > $treshold) && ($totF1F2 < $totF_akhir) &&
($hasil['S_Status']=='sukses_shutdown'))
{
	echo " Total Usage CPU $host[1] ($tot_CPU_F[1]%) melebihi treshold
($treshold%)";
	if ($hasil['S_Status']=='sukses_shutdown' ||
$hasil['S_Status']!='sukses_up')
	{
	mysql_query("UPDATE schedule SET
S_Jadwal='$waktuReal[1]',S_Status='pending_up' WHERE
S_Status='sukses_shutdown'");
	}
}
elseif (($tot_CPU_F[1]) > $treshold) //jika nilai total melebihi
threshold atas, maka server akan dihidupkan kembali
{
	echo " Total Usage CPU $host[1] ($tot_CPU_F[1]%) melebihi treshold
($treshold%)";
	if ($hasil['S_Status']=='sukses_shutdown' ||
$hasil['S_Status']!='sukses_up')
	{
	mysql_query("UPDATE schedule SET
S_Jadwal='$jam_up;',S_Status='pending_up' WHERE
S_Status='sukses_shutdown'");
	}
}
//Akhir menghidupkan kembali server
//////////////////////////////////////////////////////////////////

//Migrasi setelah server Up
//////////////////////////////////////////////////////////////////
if ($hasil['S_Status']=='sukses_up')
{
	//untuk mengurutkan nilai terbesar saja
	$query1 = mysql_query ("SELECT DISTINCT Mac_Address FROM forecast
ORDER BY F_forecast ASC LIMIT 3");
	while ($resultnya1=mysql_fetch_array($query1))
	{
		$query = mysql_query ("SELECT * FROM forecast WHERE
Mac_Address='$resultnya1[Mac_Address]' ORDER by F_id2 DESC LIMIT 1");
		while ($resultnya=mysql_fetch_array($query))
		{
			if ($resultnya['Mac_Address']=='36:66:34:32:32:38')
$no_vm='104';
			elseif
($resultnya['Mac_Address']=='62:35:34:36:30:31') $no_vm='105';
			elseif
($resultnya['Mac_Address']=='66:34:33:64:31:31') $no_vm='106';
```

```
                $no_vmnya[]=$no_vm;
                $cpu_vm[]=$resultnya['F_forecast'];
            }
        }
        echo "Vm ".$no_vmnya[0]."(".$cpu_vm[0].") akan dipindahkan";

        $vmnya= explode(' ',$vm[1]);
        for ($i=0; $i<count($vmnya); $i++)
        {
            if ($vmnya[$i]==104) $mac_vm='36:66:34:32:32:38';
            if ($vmnya[$i]==105) $mac_vm='62:35:34:36:30:31';
            if ($vmnya[$i]==106) $mac_vm='66:34:33:64:31:31';
        }

        if ($cpu_vm[0] <= $cpu_vm[1])
        {
                mysql_query("UPDATE schedule SET
S_Host_Awal='$host[1]',S_Action='Migrate_vm',S_Vm='$no_vmnya[0]',
                S_Host_Pindah='$host[0]',S_Status='pending_migrate2'
WHERE S_Status='sukses_up'");
        }
        else
        {
                mysql_query("UPDATE schedule SET
S_Host_Awal='$host[1]',S_Action='Migrate_vm',S_Vm='$no_vmnya[1]',
                S_Host_Pindah='$host[0]',S_Status='pending_migrate2'
WHERE S_Status='sukses_up'");
        }
}
//Akhir migrasi setelah server Up
//////////////////////////////////////////////////////////////////
?>
```

Eksekusi_migrate.php

```
<?php
echo '<head>
<meta http-equiv="refresh" content="10">
</head>';
include "../koneksi.php";
include('Net/SSH2.php');
date_default_timezone_set("Asia/Jakarta");
$query = mysql_query ("SELECT * FROM schedule");
$hasil=mysql_fetch_array($query);

if ($hasil['S_Host_Awal']=='Pve 1') $host='192.168.1.131';
elseif ($hasil['S_Host_Awal']=='Pve 2') $host='192.168.1.132';

if ($hasil['S_Host_Pindah']=='Pve 1') $host_pindah ='pve1';
elseif ($hasil['S_Host_Pindah']=='Pve 2') $host_pindah ='pve2';

if ($hasil['S_Status']=='pending_migrate')
{
	if ($hasil['S_Jadwal']<=date("Y-m-d H:i:00"))
	{
		$ssh = new Net_SSH2($host);
		if (!$ssh->login('root', 'komputer'))
		{
			exit('Login Failed');
		}
		else
		{
			echo "- Login sukses</br>";
			if ($ssh->exec('qm list'))
			{
				echo "- Vm tersedia</br>";
				echo $ssh->exec("qm unlock $hasil[S_Vm]");
				if (!$ssh->exec("qm migrate $hasil[S_Vm]
$host_pindah --online")) //jika berhasil tidak terdeteksi apa2, maka cek
jika berhasil maka artinya migrasi berhasil
				{
					echo "Berhasil migrasi";
					mysql_query("UPDATE schedule SET
S_Status='sukses_migrate' WHERE S_Status='pending_migrate'");

					mysql_query("INSERT INTO
history(waktu,history)
					VALUES ('".date('Y-m-d
H:i:00')."','Berhasil migrasi Vm $hasil[S_Vm] dari host
$hasil[S_Host_Awal] ke host $hasil[S_Host_Pindah]')");
				}
			}
			else echo "- Vm tidak ada</br>";
		}
	}
	else echo "Jadwal migrasi pada vm ".$hasil['S_Vm']." dari host
".$hasil['S_Host_Awal']." ke ".$host_pindah." pada ".$hasil['S_Jadwal'];
}
elseif ($hasil['S_Status']=='pending_migrate2')
```

```
{
	if ($hasil['S_Jadwal']<=date("Y-m-d H:i:00"))
	{
		$ssh = new Net_SSH2($host);
		if (!$ssh->login('root', 'komputer'))
		{
			exit('Login Failed');
		}
		else
		{
			echo "- Login sukses 2</br>";
			if ($ssh->exec('qm list'))
			{
				echo "- Vm tersedia 2</br>";
				echo $ssh->exec("qm unlock $hasil[S_Vm]");
				if (!$ssh->exec("qm migrate $hasil[S_Vm]
$host_pindah --online")) //jika berhasil tidak terdeteksi apa2, maka cek
jika berhasil maka artinya migrasi berhasil
				{
					echo "Berhasil migrasi 2";
					mysql_query("UPDATE schedule SET
S_Status='selesai' WHERE S_Status='pending_migrate2'");
					mysql_query("INSERT INTO
history(waktu,history)
					VALUES ('".date('Y-m-d
H:i:00')."','Berhasil migrasi2 Vm $hasil[S_Vm] dari host
$hasil[S_Host_Awal] ke host $hasil[S_Host_Pindah]')");
				}
			}
			else echo "- Vm tidak ada 2</br>";
		}
	}
	else echo "Jadwal migrasi pada vm ".$hasil['S_Vm']." dari host
".$hasil['S_Host_Awal']." ke ".$host_pindah." pada ".$hasil['S_Jadwal'];
}
else echo "Tidak ada jadwal migrasi pada saat ini";
?>
```

Eksekusi_migrate.php

```
<?php
echo '<head>
<meta http-equiv="refresh" content="10">
</head>';
include "../koneksi.php";
date_default_timezone_set("Asia/Jakarta");

//define('NET_SSH2_LOGGING', NET_SSH2_LOG_COMPLEX);
$query = mysql_query ("SELECT * FROM schedule");
$hasil=mysql_fetch_array($query);

if ($hasil['S_Host_Awal']=='Pve 1') $host='192.168.1.131';
elseif ($hasil['S_Host_Awal']=='Pve 2') $host='192.168.1.132';
//if ($hasil['S_Host_Awal']='Pve 3') $host='192.168.1.133';

if ($hasil['S_Host_Pindah']=='Pve 1') $host_pindah ='pve1';
elseif ($hasil['S_Host_Pindah']=='Pve 2') $host_pindah ='pve2';
//echo $hasil['S_Host_Awal'];
$result = exec("/bin/ping -c 1 -W 1 ".$host);

if ($hasil['S_Status']=='pending_shutdown')
{
	if ($result)
	{
		$ssh = new Net_SSH2($host);
		if (!$ssh->login('root', 'komputer'))
		{
			exit('Login Failed');
		}
		else
		{
			echo "teeeee";
			//echo $ssh->exec("shutdown -P now");
			//echo $ssh->getLog();
			if ($hasil['S_Host_Awal']=='Pve 1')
			{
				$ch =
curl_init("http://192.168.1.174/tesis/down_server_proxmox_1.php");
				curl_exec($ch);
			}
			elseif ($hasil['S_Host_Awal']=='Pve 2')
			{
				$ch =
curl_init("http://192.168.1.174/tesis/down_server_proxmox_2.php");
				curl_exec($ch);
			}
		}
	}
	elseif (!$result)
	{
			echo "server $hasil[S_Host_Awal] ($host) Down";
			mysql_query("UPDATE schedule SET
S_Status='sukses_shutdown'");
```

```
                    mysql_query("INSERT INTO history(waktu,history)
                    VALUES ('".date('Y-m-d H:i:00')."','Berhasil shutdown
host $hasil[S_Host_Awal]')");
       }
}
else echo "belum ada jadwal shutdown";
?>
```

Eksekusi_up.php

```
<?php
echo '<head>
<meta http-equiv="refresh" content="10">
</head>';
include "../koneksi.php";
date_default_timezone_set("Asia/Jakarta");
$query = mysql_query ("SELECT * FROM schedule");
$hasil=mysql_fetch_array($query);

if ($hasil['S_Host_Awal']=='Pve 1') $host='192.168.1.131';
elseif ($hasil['S_Host_Awal']=='Pve 2') $host='192.168.1.132';
//if ($hasil['S_Host_Awal']='Pve 3') $host='192.168.1.133';

if ($hasil['S_Host_Pindah']=='Pve 1') $host_pindah ='pve1';
elseif ($hasil['S_Host_Pindah']=='Pve 2') $host_pindah ='pve2';

$date = date_create($hasil['S_Jadwal']);
date_add($date, date_interval_create_from_date_string('+0 minutes'));
$jam_akhir=date_format($date, 'Y-m-d H:i:s');


echo "Jadwal Up :".$jam_akhir."</br>";
echo "Jam saat ini :". date("Y-m-d H:i:00")."</br>";
echo $hasil['S_Host_Awal'];

if ($hasil['S_Status']=='pending_up' && $jam_akhir<=date("Y-m-d H:i:00")
&& $hasil['S_Host_Awal']=='Pve 1')
{
       echo "tytyty";
       $ch =
curl_init("http://192.168.1.174/tesis/up_server_proxmox_1.php");
       curl_exec($ch);

       $result = exec("/bin/ping -c 1 -W 1 ".$host);
       if ($result)
       {
       mysql_query("UPDATE schedule SET S_Status='sukses_up' WHERE
S_Status='pending_up'");
       mysql_query("INSERT INTO history(waktu,history)
                           VALUES ('".date('Y-m-d H:i:00')."','Berhasil
menghidupkan kembali host $hasil[S_Host_Awal]')");
       }
}
elseif ($hasil['S_Status']=='pending_up'  && $jam_akhir<=date("Y-m-d
H:i:00") && $hasil['S_Host_Awal']=='Pve 2')
{
       echo "ffffff";
       $ch =
curl_init("http://192.168.1.174/tesis/up_server_proxmox_2.php");
       curl_exec($ch);

       $result = exec("/bin/ping -c 1 -W 1 ".$host);
       if ($result)
       {
```

```
       mysql_query("UPDATE schedule SET S_Status='sukses_up' WHERE
S_Status='pending_up'");
       mysql_query("INSERT INTO history(waktu,history)
                           VALUES ('".date('Y-m-d H:i:00')."','Berhasil
menghidupkan kembali host $hasil[S_Host_Awal]')");
       }
}
else echo "belum ada jadwal up </br>";
?>
```

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil penelitian pada tesis ini yang dimulai dari studi literatur sampai uji coba penelitian, maka dapat disimpulkan beberapa poin seperti di bawah ini:

1. Optimasi daya dengan metode prediksi pada sistem ini dapat dilakukan pada migrasi VM dan DNS, dari hasil pengujian menunjukkan bahwa dengan metode prediksi dapat mengurangi konsumsi daya sebesar 1,14 Watt dibandingkan dengan metode konvensional
2. Konsumsi daya pada pola beban meningkat adalah 13,25 Watt, dan pada pola beban menurun adalah 11,57 Watt
3. Hasil *forecast* atau prediksi usage CPU pada sistem ini dapat dilakukan menggunakan *moving average*.

5.2 Saran

Adapun saran untuk pengembangan penelitian ini pada waktu yang akan datang adalah sebagai berikut:

1. Penggunan prediksi atau forecast pada usage CPU dapat dikembangkan dengan penambahan parameter pada *moving average* dengan mempertimbangkan pola data awal.
2. Pengujian dapat dikembangkan dengan beban data yang lebih bervariasi

DAFTAR PUSTAKA

Calheiros, R., Masoumi, E., Ranjan, R. and Buyya, R. (2014), "Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS", *IEEE Transactions on Cloud Computing*, Vol. XX No. c, pp. 1–1.

Choi, K.C.K., Soma, R. and Pedram, M. (2004), "Dynamic Voltage and Frequency Scaling Based on Workload Decomposition", *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, available at:http://doi.org/10.1109/LPE.2004.1349330.

Fathurahman, M., Telekomunikasi, T., Elektro, J.T. and Jakarta, P.N. (2012), "Efisiensi Kinerja Pengelolaan Energi pada Arsitektur Data Center Komputasi Awan Menggunakan Greencloud", *JURNAL ILMIAH ELITE ELEKTRO*, Vol. 3 No. 1, pp. 6–14.

Filani, D., He, J., Gao, S., Rajappa, M., Kumar, A., Shah, P. and Nagappan, R. (2008), "Dynamic Data Center Power Management Trends, Issues, and Solutions", *Intel Technology Journal*, Vol. 12 No. 01, pp. 59–67.

Hewlett-Packard, Intel, Microsoft, Phoenix Technologies Ltd and Toshiba. (2011), *Advanced Configuration and Power Interface Specification*.

Ho Lee, S. and Keun Oh, J. (2004), "Method for Measuring Quantity of Usage of CPU", available at: http://www.google.com/patents/US6804630 (accessed 30 April 2015).

Ikram, M., Babar, Q.U.A., Anwar, Z. and Malik, A.W. (2013), "GSAN: Green cloud-simulation for storage area networks", *Proceedings - 11th International Conference on Frontiers of Information Technology, FIT 2013*, pp. 265–270.

Intel. (2003), "Intel® 80200 Processor based on Intel ®XScale™ Microarchitecture", No. January, available at: http://int.xscale-freak.com/XSDoc/IOP303/27341405.pdf.

Jati Waloeyo, Y. (2012), *Cloud Computing*, edited by Yesaya Jati, P., Penerbit Andi dan Elcom, Yogyakarta.

Kliazovich, D., Bouvry, P. and Khan, S.U. (2012), "GreenCloud : A Packet-level Simulator of Energy- aware Cloud Computing Data Centers", *Journal of Supercomputing*, Vol. 62 No. 3, pp. 1263–1283.

Lakshmanan, K., Kato, S. and Rajkumar, R. (Raj). (2010), "Scheduling Parallel Real-Time Tasks on Multi-core Processors", *2010 31st IEEE Real-Time Systems Symposium*, pp. 259–268.

Lee, W.Y. (2009), "Energy-saving DVFS Scheduling of Multiple Periodic Real-time Tasks on Multi-core Processors", *Proceedings - IEEE International Symposium on Distributed Simulation and Real-Time Applications, DS-RT*, pp. 216–223.

Lorido-Botran, T., Miguel-Alonso, J. and A. Lozano, J. (2012), *Auto-Scaling Techniques for Elastic Applications in Cloud Environments*.

Mahadevan, P., Sharma, P., Banerjee, S. and Ranganathan, P. (2009), "Energy

Aware Network Operations", *IEEE INFOCOM Workshops 2009*, pp. 1–6.

Mell, P. and Grance, T. (2011), "The NIST Definition of Cloud Computing", *NIST Special Publication*, Vol. 145, p. 7.

Roy, N., Dubey, A. and Gokhale, A. (2011), "Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting", *2011 IEEE 4th International Conference on Cloud Computing*, Ieee, pp. 500–507.

Sofana, I. (2012), *Cloud Computing Teori Dan Praktik*, Informatika, Bandung.

Sueur, E. Le and Heiser, G. (2010), "Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns", *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, pp. 1–8.

"TIA-942 Data Centre Standards Overview". (2008), .

Wahyu W, P., Harsono, T.B. and Annggis S, N. (2014), "Analisis Performansi Sistem Dari Implementasi Auto-Scaling pada Cloud Computing Dengan Menggunakan Predictive System Simple Moving Average", Bandung, pp. 1–7.

Yaffee, R.A. and McGee, M. (2000), *Introduction to Time Series Analysis and Forecasting: With Applications of SAS and SPSS*, Academic Press, New York.

Yang, J., Liu, C., Shang, Y., Mao, Z. and Chen, J. (2013), "Workload Predicting-Based Automatic Scaling in Service Clouds", available at:http://doi.org/10.1109/CLOUD.2013.146.

Zulkarnain, I. (2012), "Akurasi Grafik Main Chart Dalam Prediksi Harga Saham Harian : Kasus The Winnest Dan The Losest", *Jurnal Ilmiah STIE MDP*, Vol. 1 No. 2, pp. 74–83.

BIOGRAFI

Amirullah, terlahir di sebuah daerah ujung barat Indonesia, tepatnya di Lhokseumawe daerah Naggroe Aceh Darussalam pada 28 Agustus 1989. Penulis telah menerima gelar sarjana dibidang Teknik Informatika dari kampus Politeknik Negeri Lhokseumawe dengan titel SST pada tahun 2012. Saat menempuh kuliah D4 di Politeknik Negeri Lhokseumawe, penulis juga aktif di bidang kemahasiswaan dan sangat tertarik menekuni dan mengembangkan ilmu di bidang *networking.* Kemudian pada tahun 2013, penulis melanjutkan studi ke perguruan tinggi Institut Teknologi Sepuluh Nopember (ITS) pada program Pascasarjana jurusan Teknik Informatika. Dengan menyelesaikan kuliah di program Pascasarjana ITS ini, penulis berhasil mendapatkan tambahan gelar Master di bidang Teknik Informatika.

Contact : amir.pnl08@gmail.com