IMPLEMENTASI *DISTRIBUTED BARTERING ALGORITHM* PADA *LIVE MIGRATION* MESIN VIRTUAL

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Meliza Gratia Dorothy Latuputty
NIM: 125150209111026

PROGRAM STUDI INFORMATIKA/ILMU KOMPUTER
PROGRAM TEKNOLOGI INFORMASI DAN ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2016

PENGESAHAN

IMPLEMENTASI *DISTRIBUTED BARTERING ALGORITHM*
PADA *LIVE MIGRATION* MESIN VIRTUAL

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
Meliza Gratia Dorothy Latuputty
NIM. 125150209111026

Skripsi ini telah diuji dan dinyatakan lulus pada

26 Agustus 2016

Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Sabriansyah Rizqika Akbar, S.T, M.Eng	Achmad Basuki, S.T, M.MG, Ph.D
NIP. 19820809 201212 1 004	NIP. 19741118 200312 1 002

Mengetahui, Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP. 19710518 200312 1 001

PERNYATAAN ORISINALITAS SKRIPSI

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah SKRIPSI ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk untuk memperoleh gelar akademik di suatu perguruan tinggi dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis dikutip dalam naskah ini dan disebutkan dalam sumber kutipan dan daftar pustaka.

Apabila ternyata di dalam naskah SKRIPSI ini dapat dibuktikan terdapat unsur-unsur PLAGIASI, saya bersedia SKRIPSI ini digugurkan dan gelar akademik yang telah saya peroleh (SARJANA) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70)

Malang, 1 September 2016

Meliza Gratia Dorothy Latuputty
125150209111026

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadirat Tuhan yang Maha Esa, yang telah melimpahkan berkat dan anugerah-Nya kepada penulis sehingga penulis dapat menyelesaikan skripsi ini yang berjudul "**IMPLEMENTASI *DISTRIBUTED BARTERING ALGORITHM* PADA *LIVE MIGRATION* MESIN VIRTUAL**"

Pada penyusunan skripsi ini tidak semata-mata hasil kerja penulis sendiri, melainkan juga berkat dan bimbingan dan dorongan dari pihak-pihak yang telah membantu, baik secara materi maupun non materi. Maka dari itu penulis ingin mengucapkan banyak terima kasih yang tak terhingga serta penghargaan yang setinggi-tingginya kepada orang-orang yang telah membantu penulis secara langsung maupun tidak langsung kepada yang terhormat:

1. Kedua orang tua penulis yang telah memberikan doa, nasehat dan kasih sayang dan kesabarannya dalam membesarkan dan mendidik penulis
2. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D selaku Ketua Jurusan Informatika yang telah memberikan kesempatan kepada penulis untuk menyelesaikan skripsi ini.
3. Bapak Sabriansyah Rizqika Akbar. S.T, M.Eng, dan Bapak Achmad Basuki, S.T, M.MG, Ph.D. selaku dosen pembimbing skripsi yang telah dengan sabar membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini.
4. Segenap bapak dan ibu dosen program studi Informatika/Ilmu Komputer beserta seluruh staff administrasi yang telah membantu selama perkuliahan.
5. Teman-teman Informatika/Ilmu Komputer angkatan SAP 2012 yang telah memberikan masukan dan inspirasi kepada penulis selama menempuh studi dan menyelesaikan skripsi ini.

Penulis menyadari bahwa penulisan skripsi ini jauh dari kata sempurna. Oleh karena itu, penulis mengharapkan masukan berupa saran dan kritik dari semua pihak demi tercapainya kesempurnan dalam skripsi ini. Akhir kata semoga penulisan skripsi ini dapat memberikan manfaat bagi semua pihak.

Malang, 1 September 2016

Penulis

meliza.latuputty29@gmail.com

ABSTRAK

Munculnya *cloud computing* telah menyebabkan perkembangan minat dalam penyebaran dan penggunaan berbagai macam aplikasi pada lingkungan komputasi. Keberhasilan *cloud computing* didorong dengan adanya penggunaan virtualisasi sebagai teknologi yang mendasari mereka. Namun, dalam beberapa aplikasi, perhitungan analisis ilmiah dan data yang memiliki pola yang rumit dari komunikasi sehingga membutuhkan pertukaran data dalam jumlah besar untuk melaksanakan perhitungan yang akan dilakukan. Untuk aplikasi tersebut, pola komunikasi antar komponen yang berbeda merupakan faktor kunci yang harus diperhatikan selama pengalokasian sumber daya. Selain itu, sering ditemukan *bandwidth* jaringan menjadi sumber kemacetan dibanyak *platform* virtualisasi yang ada. Berdasarkan permasalahan tersebut, dilakukan percobaan proses komunikasi yang dapat mengusulkan perpindahan mesin virtual ke *server* lain. Penulis menggunakan *distributed bartering algorithm* dimana *server* fisik secara independen bernegosiasi untuk penempatan *virtual machine* (VM) atas dasar informasi lokal, berdasarkan gangguan afinitas yang berbeda antar pasangan VM, dan negosiasi penempatan yang lebih baik untuk VM. Jika proses negosiasi selesai maka dilakukan migrasi VM yang akan saling bertukar *volume* pada trafik jaringan yang lebih dekat satu sama lain. Hasil pengujian menunjukkan proses pengimplementasian *distributed bartering algorithm* dapat berjalan dengan baik. *Distributed bartering algorithm* dapat menjadi salah satu solusi untuk mengurangi kemacetan dan *overhead* jaringan. Hasil pengujian memperlihatkan bahwa rata-rata dari tiap skenario tidak melebihi ambang batas yaitu 400 Mbits/sec sehingga hal ini menunjukan bahwa metode yang diusulkan mampu menyeimbangkan beban trafik data antar HVM dan meminimalisir terjadi kemacetan dan *overhead* jaringan. Metode negosiasi dan *swapping* antar HVM juga dapat diterapkan dengan baik dan mampu menyeimbangkan kelebihan beban yang terjadi antar HVM.

Kata kunci: Virtualisasi, *Distributed Bartering Algoritmh, Live Migration*

ABSTRACT

The emergence of cloud computing has led to the development of interest in the deployment and the use of a wide range of applications in the computing environment. The success of cloud computing is driven by their use of virtualization as their underlying technology. However, in some applications, scientific analysis and calculation of data which have complex patterns of communication data requires large amounts of data to perform calculations to be performed. For such applications, the pattern of communication between the different components is a key factor that must be considered during the allocation of resources. Additionally, it is often found that network bandwidth become source of bottlenecks in many existing virtualization platform. Based on these problems, the authors conducted communication process experiments to propose a transfer of virtual machines to other servers. The author uses a distributed bartering algorithm where physical server independently negotiate for VM placement on the basis of local information, based on the different affinity interference between the pair VM, and negotiating better placement for the virtual machine (VM). If the negotiation process is completed then performed VM migration that would exchange traffic volume on the network closer to each other. The test results showed that distributes bartering algorithm implementation can run well. Distributed bartering algorithm can be one solution to reduce congestion and network overhead. The test results also showed that the average of each scenario did not exceed the threshold of 400 Mbits/sec so this shows that the proposed method is able to balance the load of data traffic occurs between HVM and minimize congestion and network overhead. Negotiation and swapping between HVM also be applied properly and are able to balance the overload that occurs between HVM.

Keywords: *Virtualisation, Distributed Bartering Algorithm, VM Migration*

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Saat ini perkembangan teknologi informasi memegang peranan penting pada kehidupan manusia. Seiring dengan pesatnya pembangunan yang ditandai dengan semakin canggihnya arsitektur komputer saat ini, teknologi internet dapat dikembangkan menjadi komputasi awan. Munculnya komputasi awan telah menyebabkan perkembangan minat dalam penyebaran dan penggunaan berbagai macam aplikasi pada lingkungan komputasi. Dengan komputasi awan, pengguna dapat mengakses semua aplikasi dan dokumen dari tempat manapun dan menggunakan *gadget* apapun. Komputasi awan juga menyajikan tren teknologi yang signifikan, dan secara jelas membentuk kembali proses teknologi informasi dan pasar IT (Furth, 2010).

Keberhasilan *cloud* telah didorong dengan adanya penggunaan virtualisasi sebagai teknologi yang mendasari mereka. Virtualisasi merupakan fitur kunci dari komputasi awan yang memungkinkan beberapa mesin virtual untuk berbagi sumber daya pada mesin fisik tunggal yang dapat meningkatkan pemanfaatan sumber daya data center (Nilesh & Rajesh, 2013). Mesin virtual (VM) memberikan fleksibilitas dan mobilitas melalui proses migrasi yang mudah, yang memungkinkan pemetaan yang dinamis pada mesin virtual (VM) ke sumber daya yang tersedia. Mesin virtual (VM) juga menyediakan isolasi dan keamanan kinerja yang memfasilitasi *multiplexing* dan pemanfaatan pembagian sumber daya. Namun, dalam beberapa aplikasi perhitungan analisis ilmiah dan data yang memiliki pola yang rumit pada komunikasi sehingga membutuhkan pertukaran data dalam jumlah besar untuk melaksanakan perhitungan yang akan dilakukan. Untuk aplikasi tersebut, pola komunikasi antar komponen yang berbeda merupakan faktor kunci yang harus diperhatikan selama pengalokasian sumber daya. Selain itu, performa jaringan menjadi lebih lambat karena disebabkan oleh banyaknya paket pada jaringan, dimana trafik data melebihi dari kapasitas *bandwidth* yang ada (Armbrust, 2009).

Akibat dari permasalahan itu, penyedia *cloud* berusaha untuk mengurangi *overhead* jaringan sebanyak mungkin. Selain mempertimbangkan karakter fisik (kecepatan CPU, memori dan penyimpanan) dari *node* yang ada pada mesin virtual yang ditempatkan, topologi jaringan juga harus menjadi pertimbangan dalam meningkatkan efisiensi platform dan mengurangi permasalahan jaringan.

Berdasarkan permasalahan tersebut, dilakukan proses komunikasi yang dapat mengusulkan perpidahan mesin virtual ke *server* lain. Penulis menggunakan *distributed bartering algorithm* yang dimana *server* fisik secara independen bernegosiasi untuk penempatan VM atas dasar informasi lokal, berdasarkan gangguan afinitas yang berbeda antar pasangan VM, negosiasi penempatan yang lebih baik untuk VM. Jika proses negosiasi selesai maka dilakukan migrasi VM yang

dimana akan saling bertukar *volume* pada trafik jaringan yang lebih dekat satu sama lain.

Harapan yang ingin dicapai dari penelitian ini adalah penerapan *distributed bartering algorithm* dapat berhasil dengan baik dan dapat melakukan migrasi serta proses negosiasi penempatan VM juga dapat berjalan dengan optimal sehingga dapat mengatasi permasalahan biaya komunikasi yang tinggi pada *server* fisik.

1.2 Rumusan Masalah

Berdasarkan pada permasalahan yang diangkat, maka dapat dirumuskan masalah untuk penelitian ini adalah sebagai berikut:

1. Bagaimana menerapkan *distributed bartering algorithm* pada *live migration* mesin virtual?
2. Bagaimana metode negosiasi dan *swapping* melakukan optimasi *live migration* mesin virtual pada lingkungan komputasi awan?

1.3 Tujuan

Tujuan yang ingin dicapai dalam penelitian ini adalah menerapkan *distributed bartering algorithm* pada proses *live migration* sehingga dapat mengurangi kemacetan jaringan.

1.4 Manfaat

Pada penelitian ini proses migrasi menggunakan *distributed bartering algorithm* diharapkan dapat membantu proses negosiasi penempatan VM sehingga VM dapat bertukar besar *volume* lalu lintas jaringan yang lebih dekat satu dengan lain.

1.5 Batasan Masalah

Pembatasan masalah penelitian ini adalah sebagai berikut.

1. Semua komputer yang digunakan dalam penelitian memiliki spesifikasi yang sama dan sistem dibangun pada segmentasi jaringan lokal.
2. Sumber daya komputasi hanya fokus pada penggunaan *bandwidth*.
3. Semua mesin virtual yang dibangun berbagi satu media penyimpanan yang sama.
4. Pembuatan program perangkat lunak tambahan menggunakan *Bash shell.*
5. Pengujian sistem dilakukan dengan melihat trafik antar VM.

1.6 Sistematika Pembahasan

Penyusunan penelitian ini nantinya disusun dengan sistematika pembahasan sebagai berikut:

BAB 1 PENDAHULUAN

Bab pendahuluan terdiri dari latar belakang masalah, batasan masalah, rumusan masalah, tujuan penelitian, manfaat penelitian, dan sistematika pembahasan penelitian. Bab ini menjadi dasar dari keseluruhan pelaksanaan penelitian ini.

BAB 2 LANDASAN KEPUSTAKAAN

Bab landasan kepustakaan membahas mengenai teori-teori yang berkaitan dan menunjang dalam penyelesaian penelitian ini. Teori-teori tersebut diambil dari beberapa jurnal, buku, dan sumber referensi lainnya.

BAB 3 METODOLOGI

Bab metode penelitian menguraikan tentang metodologi dalam membangun sistem. Metodologi yang dimaksudkan adalah langkah-langkah seperti perancangan, implementasi, pengujian, dan analisis sistem secara umum dalam bentuk diagram alir.

BAB 4 PERANCANGAN DAN IMPLEMENTASI

Bab perancangan membahas tentang proses perancangan sistem yang akan dibangun. Perancangan tersebut dilakukan dengan memperhatikan kebutuhan perangkat lunak dan perangkat keras yang telah ditentukan pada tahap analisis kebutuhan sebelumnya. Pada bab implementasi, membahas dan menjelaskan pembuatan sistem secara keseluruhan dengan menampilkan beberapa gambar dari hasil implementasi yang telah dilakukan.

BAB 5 PENGUJIAN DAN ANALISIS

Pada bab ini berisi tentang pengujian dan analisis terhadap sistem yang dibangun. Pengujian dan analisis akan dijelaskan dalam bentuk gambar-gambar dan grafik dari hasil proses pengujian dan analisis sistem.

BAB 6 PENUTUP

Bab penutup menguraikan tentang kesimpulan dari keseluruhan pelaksanaan penelitian serta diberikan saran-saran guna perbaikan dan penyempurnaan sistem yang telah dibangun.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Untuk penelitian yang terkait dengan pengoptimalan kinerja *live migration* mesin salah satunya telah dilakukan oleh Aditya Angga (2012) dengan judul "*Optimasi Penggunaan Sumber Daya Komputasi Di Lingkungan Komputasi Awan*". Dalam penelitian tersebut penulis menjabarkan bahwa sistem dibangun dengan menggunakan metode *load balancing* untuk melakukan penyebaran beban kerja dengan memindahkan mesin virtual pada *server* yang memiliki beban kerja CPU tinggi ke *server* yang memiliki beban kerja CPU rendah. Sedangkan metode *load aggregation* digunakan jika sistem mendeteksi penurunan beban kerja CPU dalam satu atau lebih *server* fisik, beberapa mesin virtual akan digabungkan kembali ke dalam sebuah *server* fisik virtualisasi. Berdasarkan penelitian tersebut, sistem yang dibangun dapat mengoptimalkan pengunaan sumber daya komputasi dan menyeimbangkan beban kerja CPU dalam semua *server* fisik.

Penelitian terkait lainnya mengenai pengoptimalan *live migration* salah satunya dilakukan oleh Sonnek Jason (2010) dengan judul "Starling: *Minimizing Communication Overhead in Virtualized Computing Platforms Using Decentralized Affinity-Aware Migration".* Penelitian tersebut menyajikan teknik migrasi desentralisasi *affinity-aware* yang menerapkan *distributed bartering algorithm* pada proses *live migration* mesin virtual secara dinamis menyesuaikan penempatan VM sehingga biaya *overhead* komunikasi dapat diminimalkan.

Dengan melihat beberapa penelitian yang telah dilakukan sebelumnya maka penelitian ini berfokus tentang bagaimana cara untuk melakukan optimasi *live migration* mesin virtual dengan menggunakan *distributed bartering algorithm*.

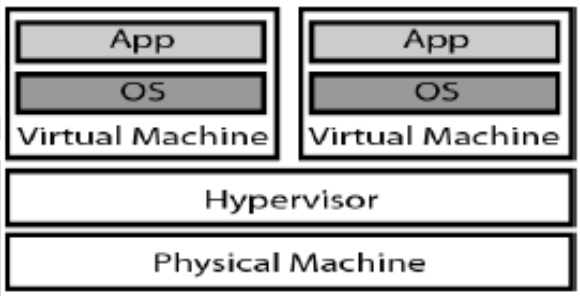
2.2 Dasar Teori

2.2.1 Virtualisasi

Virtualisasi adalah teknologi arsitektur komputer dengan yang beberapa mesin virtual (VM) *multiplexing* dalam mesin *hardware* yang sama. Tujuan dari VM adalah untuk meningkatkan berbagi sumber daya oleh banyak pengguna dan meningkatkan kinerja komputer dalam hal pemanfaatan sumber daya dan fleksibilitas aplikasi. *Hardware* sumber daya (CPU, memori, I/O perangkat, dll) atau sumber daya perangkat lunak (sistem operasi dan perpustakaan *software*) dapat virtual di berbagai lapisan fungsional (Hwang, 2012). Sederhananya, virtualisasi adalah emulasi perangkat keras dalam sebuah *platform* perangkat lunak. Hal ini memungkinkan satu komputer untuk mengambil peran beberapa komputer. Pada dasarnya, teknik virtualisasi ini memungkinkan sebuah komputer untuk melakukan pekerjaan dari beberapa komputer, dengan berbagi sumber daya dari perangkat keras tunggal di beberapa lingkungan (Myint, 2010).

Pada lingkungan komputasi awan, sebagian besar penyedia layanan *cloud* menggunakan teknik virtualisasi. Hal ini dikarenakan dengan menggunakan

teknologi virtualisasi, pemanfaatan sumber daya komputasi lebih efisien dan memungkinkan beberapa penguna untuk berbagi infrastuktur fisik sama serta sumber data yang lainya. Selain itu, terdapat beberapa kelebihan teknik virtualisasi lainya, yaitu efektifitas perbaikan proses pemulihan kegagalan sistem, pengurangan kebutuhan konsumsi energi, kebutuhan tempat, beban jaringan, serta pengurangan biaya tes dan pengembangan perangkat lunak.



Gambar 2.1 Prinsip Virtualisasi
Sumber : Anja (2011)

2.2.2 Pendekatan Virtualisasi

Terdapat dua jenis pendekatan teknik virtualisasi, yaitu *full virtualization* dan *paravirtualization*. Penjelasan mengenai kedua teknik adalah sebagai tersebut.

a. *Full virtualization*

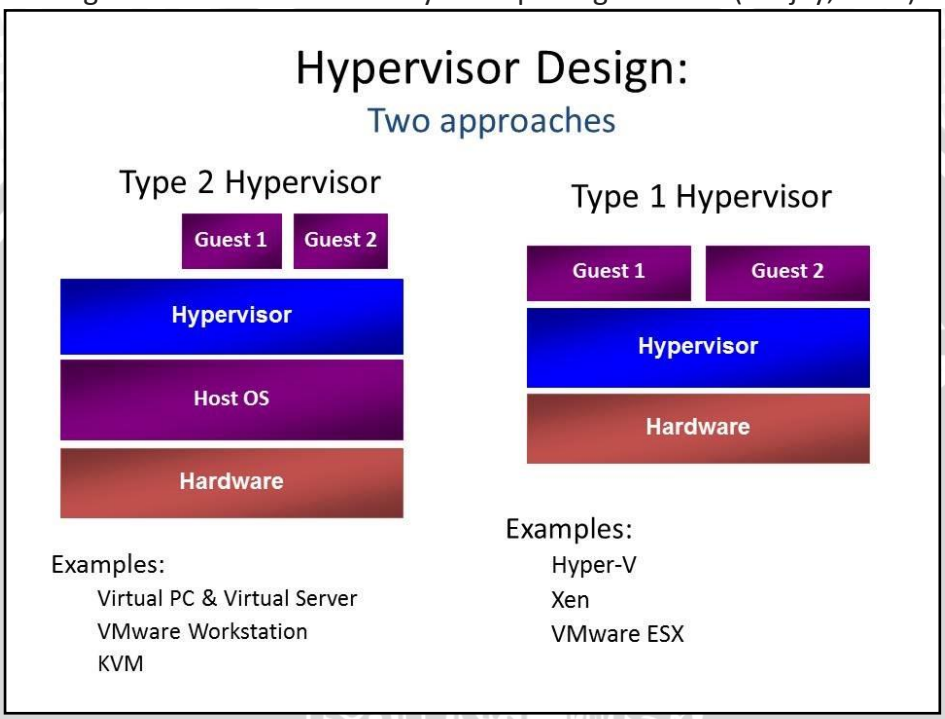
Full virtualization adalah salah satu teknik virtualisasi yang seluruh sistem komputer dibuat ke dalam kontruksi *software*, dimana konstruksi tersebut berfungsi sebagai *hardware* aslinya. *Full virtualization* digunakan untuk implementasi pada berbagai macam lingkungan mesin virtual dimana pada teknik ini disediakan simulasi lengkap dari perangkat keras sehingga memberikan keuntungan yakni semua perangkat lunak yang bisa dieksekusi langsung pada mesin virtual termasuk pada sistem operasi. Dalam *full virtualization* virtualisasi *hypervisor* menyediakan sebagian besar antarmuka perangkat keras yang sama seperti yang disediakan oleh platform fisik *hardware*. Ini berarti bahwa OS dan aplikasi yang berjalan dalam penuh virtualisasi tidak perlu dimodifikasi untuk virtualisasi untuk bekerja jika OS dan aplikasi yang kompatibel dengan *hardware* yang mendasari (Karen, 2011).
Solusi yang menerapkan *full virtualization* yaitu *Hypervisor* VM Ware, VirtualBox dari Oracle, KVM, dan QEMU.

b. *Paravirtualization*

Paravirtualization merupakan teknik virtualisasi yang efisien dan ringan diperkenalkan oleh tim Xen Proyek, kemudian diadopsi oleh solusi virtualisasi lainnya. Pengimplementasian *Paravirtualization* digunakan pada berbagai macam lingkungan mesin virtual, dengan lingkungan mesin virtual hanya disediakan simulasi perangkat keras secara sebagian. Salah satu solusi yang menerapkan *paravirtualization* yaitu Xen.

2.2.3 *Hypervisor*

Hypervisor menciptakan beberapa virtual *server* dalam satu *server* fisik. Setiap *server* virtual bisa memiliki *operating sistem* (OS) yang terinstal. Banyak *server* virtual bisa dioperasikan secara simultan dan independen dari satu sama lain. *Hypervisor* memungkinkan penyatuan prosesor dan sumber daya memori. Instalasi *hypervisor* pada *server host* memungkinkan untuk menjalankan beberapa sistem operasi secara be*RSA*maan menggunakan virtualisasi menggunakan teknik virtualisasi yang menyediakan dukungan infrastruktur untuk beberapa mesin virtual dengan virtualisasi sumber daya fisik perangkat keras (Sanjay, 2012).



Gambar 2.2 Arsitektur *Hypervisor*
Sumber: Microsoft (2011)

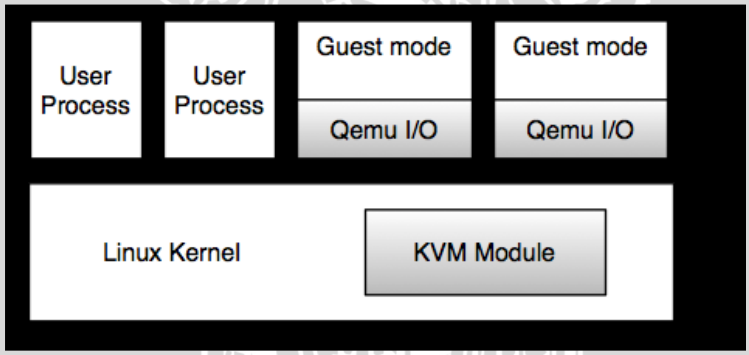
Berdasarkan gambar arsitektur *hypervisor* tersebut, *hypervisor* terdiri atas 2 tipe, yaitu *hypervisor* tipe 1 dan *hypervisor* tipe 2. *Hypervisor* tipe 1 (*native, bare metal*) mewakili teknik virtualisasi *full virtualization* sedangkan *hypervisor* tipe 2 (*hosted*) mewakili *paravirtualization.* Dalam penelitian ini, akan menerapkan teknik virtualisasi *full virtualization* dengan menggunakan *hypervisor Kernel-Based Mesin virtual* (KVM).

2.2.4 *Kernel-Based Virtual Machine* (KVM)

Kernel-Based Virtual Machine (KVM) adalah solusi virtualisasi kreatif berbasis Linux. Tidak seperti *hypervisors* umum lainnya yang melakukan dasar penjadwalan, manajemen memori sendiri, KVM menemukan kesamaan modul antara VM dan kernel sistem operasi, sehingga beralih ke kernel Linux untuk semua fungsi umum.

KVM sepenuhnya terintegrasi pada sistem operasi Linux baik sebagai *host* dan *guest*. Tidak seperti *hypervisor*s lainnya, KVM tidak membuat perbedaan antara *host* yang berjalan baik atau mode *hypervisor*. Dualitas dalam desain ini telah membantu KVM berkembang secara cepat dan menjadi stabil, highperforming *hypervisor*, dan diposisikan untuk mengungguli *hypervisor*s lain yang tersedia di pasar. Terdapat dua prinsip yang membuat KVM menjadi *hypervisor* dengan kinerja tinggi dan melampaui *open source hypervisor*s lainnya (Khoa, 2013).

1. Prinsip desain pertama meliputi:
   a. Memanfaatkan semua kemampuan virtualisasi *hardware*-dibantu disediakan oleh Intel® Virtualization Technology (VT) dan AMD® Secure Virtual Machine (SVM)
   b. Fitur ekstensi virtualisasi *hardware* terbaru.
   c. Eksploitasi kemampuan *hardware* sekaligus menjaga *overhead* virtualisasi KVM
2. Prinsip desain kedua meliputi:
   a. Memanfaatkan sistem operasi Linux
   b. Memenuhi banyak komponen yang dibutuhkan oleh *hypervisor*, seperti manajemen memori, penjadwalan, I/O *stack*, dan *device driver* dan sebagainya tidak perlu menulis dari awal sehingga lebih efisien.



Gambar 2.3 Arsitektur Virtualisasi KVM
Sumber : Timo (2011).

2.2.5 *Live migration*

Live migration merupakan salah satu fitur dari teknik virtualisasi dan merupakan objek permasalahan yang akan diangkat penulis dalam penelitian ini. Penggunaan fitur *live migration* ini adalah bentuk upaya yang dilakukan untuk melakukan penambahan sumber daya komputasi melalui ekspansi jumlah data center dengan pemindahan layanan dari *data center* yang ada ke *data center* yang baru (Al-Kiswany, 2011).

Saat proses *live migration*, klien tidak ikut terlibat dalam proses ini sehingga tetap dapat mengakses layanan secara normal tanpa merasakan adanya perpindahan *server*. Berikut merupakan urutan alogritma umum *live migration*

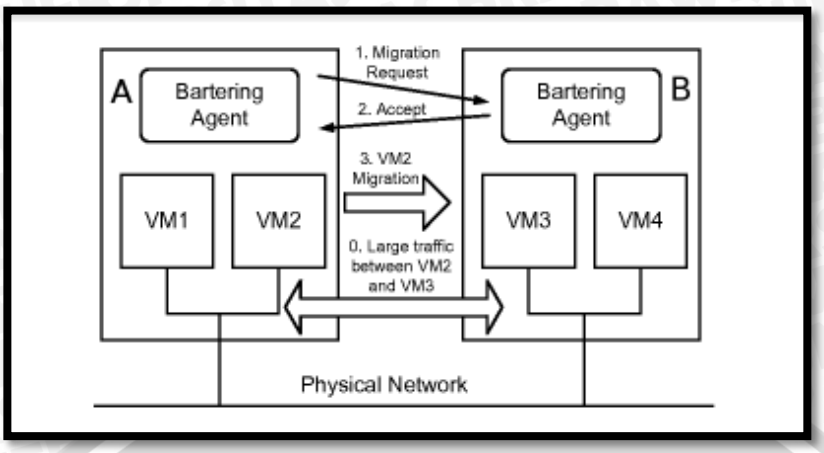
mesin virtual antara salah satu mesin fisik ke mesin fisik yang lainya, yaitu (Lublin, 2007).

1. *Request* migrasi mesin virtual diterima (migrasi mesin virtual dari mesin fisik A ke mesin fisik B)
   a. Membuat perintah-perintah eksternal.
   b. Menghubungkan dan mengirimkan header.
   c. Mengalokasikan pengaturan dan sumber daya.
2. Pemindahan *memory*
   a. Proses pertama, *memory pages* (iterasi pertama) ditransfer secara menyeluruh
   b. Untuk iterasi selanjutnya,mentransfer seluruh *dirty pages* dari iterasi i-1
   c. Proses dilakukan secara terus menerus hingga selesai
3. Mesin virtual dihentikan
4. Pemindahan *VM State*
   a. Setiap perangkat dipindahkan ke state masing-masing
   b. Pemindahan *dirty pages* dari iterasi terakhir
5. Melanjutkan mesin virtual
   a. Pada mesin B, jika proses migrasi berhasil

      Mengirimkan paket broadcast melalui e*thernet* untuk memberikan informasi lokasi mesin virtual yang baru
   b. Pada *localhost* mesin A jika proses migrasi gagal

2.2.6 *Distributed Bartering Algorithm*

Distributed Bartering Algorithm diusulkan untuk meminimalkan *overhead* dan meningkatkan skalabilitas yang memungkinkan VM untuk bernegosiasi penempatan pada *server* fisik yang lebih dekat pada data yang mereka butuhkan. Algoritma ini menghindari kemacetan dan titik pusat kegagalan yang terkait dengan solusi terpusat, dan juga dapat beradaptasi lebih cepat untuk perubahan dinamis lokal pada perubahan pola dan topologi jaringan. Selain itu pendekatan desentralisasi tersebut memungkinkan penggunaan untuk kebijakan yang berbeda dengan agen yang berbeda berdasarkan lokasi serta persyaratan aplikasi secara spesifik dari VMS yang dikelola.



Gambar 2.4 Sistem *Distributed Bartering Algorithm*
Sumber: (Sonnek, 2009)

Ketika total trafik antar mesin virtual dan *server* fisik melebihi ambang batas maka agen barter akan melakukan negosiasi untuk relokasi dengan *server* yang diinginkan dan memulai migrasi jika berhasil.

a. Negosiasi Migrasi:
   Agen barter secara berkala akan memeriksa jejak komunikasi untuk tiap *host* VM pada *server* untuk menentukan trafik yang dikirim ke mesin fisik lain melebihi trafik yang dikirim ke VM *host* lain pada *server* fisik yang sama. Jika hal ini terjadi misalnya, trafik VM 2 untuk agen barter B lebih tinggi dari trafik yang dikirim ke VM dan total *volume* trafik melebihi ambang batas migrasi, agen barter A akan mencoba untuk menegosiasikan "rumah baru" dengan mengirimkan permintaan migrasi ke *server* fisik (B). Setelah menerima permintaan migrasi, agen barter B akan mengambil salah satu tindakan berikut:
   1. Jika B mempunyai kapasitas yang tersedia untuk menjadi *host* VM 2, ia akan menerima tanggapan A. Dalam hal ini, agen barter pada A akan memulai *live migration* dari VM 2 ke B.
   2. Jika B tidak memiliki kapasitas yang tersedia, maka akan kembali ke daftar calon *swap*. Jika agen A menemukan calon *swap* yang diinginkan maka proses *swap* akan dilakukan dengan VM 2 bermigrasi ke B dan calon *swap* yang dipilih (misalnya VM 4) bermigrasi ke A.
   3. Jika tidak ada kandidat *swap* yang cocok, maka B akan mengembalikan daftar tetangga: *node* yang terdekat dalam hal *bandwidth* jaringan/*latency*. Dalam hal ini, agen barter A akan menghubungi masing-masing tetangga pada gilirannya secara rekursif memulai proses relokasi.

   Agen barter hanya akan bermigrasi VM ke tetangga jika tetangga memiliki *bandwidth* yang lebih tinggi ke *server* yang diinginkan (B) dari *host* asli (A). Jika tidak ada tetangga yang memiliki kriteria migrasi *host* VM, *server* akan menyerah dalam upaya proses migrasi.

b. VM *Swapping*:
   Untuk mengaktifkan migrasi VM 2 ke mesin B, *distributed bartering algorithm* memungkinkan untuk VM *swapping* dimana agen barter pada kedua mesin

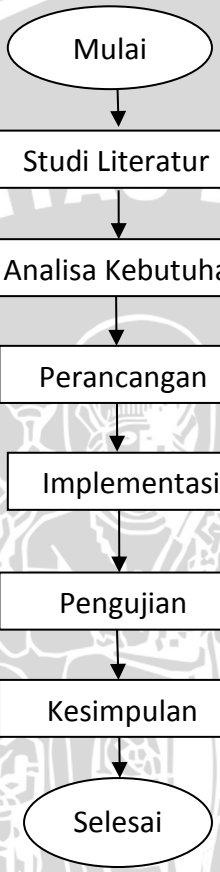
saling menukar VM untuk menghormati keterbatasan sumber daya dimasing-masing *server*. *Swapping* dilakukan hanya jika proses *swap* akan menghasilkan penempatan yang lebih baik dari VM dalam hal afinitas komunikasi mereka. Ada dua keputusan yang terlibat dalam proses *swapping*:

1. Memilih calon *swap*: ketika agen barter pada *server* penuh menerima permintaan migrasi (misalnya, mesin B dari A untuk VM 2), perlu untuk memilih satu set VM local yang memungkinkan untuk *swapping* jika ada.
   Ada 2 jenis VM yang memungkinkan untuk calon *swap*:
   a. Tipe 1: VM yang akan mendapatkan keuntungan dengan pindah ke A dari B. Ini bisa jadi trafik VM ke mesin A lebih besar dari lalu lintas yang dikirim ke VM dan mungkin pernah mencoba migrasi ke A tapi gagal sebelumnya. A juga dapat menjadi daftar tetangga seperti VM *server* yang diinginkan, sehingga dapat mengambil manfaat dari perpindahan ke A.
   b. Tipe 2: sebuah VM yang terisolasi, yaitu salah satu yang tidak berkomunikasi dengan VM lainnya. VM tidak akan terpengaruh terlepas dimana ia akan ditempatkan. Setiap agen menyimpan daftar calon pertukaran seperti yang berjalan pada mesin, serta mengirimkan daftar yang tepat tergantung dengan permintaan *server*.
2. Membuat keputusan *swap*:
   Setelah *server* menerima daftar calon *swap*, maka diperlukan untuk memilih salah satu dari daftar yang ada. Jika terdapat beberapa kandidat *swap*, maka *server* akan meminta untuk memberikan preferensi yang lebih tinggi untuk 1 calon. Pada kasus ini, juga akan memastikan bahwa pemilihan calon *swap* akan berakhir dengan trafik mesin yang lebih tinggi setelah terjadi pertukaran. Pemeriksaan ini diperlukan untuk memastikan manfaat keseluruhan dari *swap*. Misalnya dalam contoh diatas, VM 2 dapat banyak berkomunikasi dengan baik dengan VM 3 dan VM 4. Dalam hal ini VM 2 dan VM 4 tidak akan membuat perbedaan dalam trafik secara keseluruhan sementara menimbulkan *overhead* yang tidak memerlukan migrasi. Sebuah calon tipe 2 dipilih hanya jika tidak ada kandidat tipe 1. Sebagai tambahan, karena proses migrasi ini tidak dilakukan secara gratis, maka digunakan juga dengan menggunakan inersia parameter *swap* untuk menimbang biaya *swapping*. Proses *swap* dilakukan dengan memicu tiap migrasi secara berturut-turut, sehingga ketika sementara salah satu mesin akan kelebihan beban, memungkinkan *swap* untuk disinkronisasi dengan tiap VM dalam keadaan konsisten di tiap proses migrasi. Dalam beberapa kasus, penggunaan *server* secara sementara mungkin diperlukan untuk menghindari *overload server* yang berpartisipasi dalam proses *swap*.
   a. Computing Migration Threshold :
   Seperti yang sudah disebutkan di atas, untuk memastikan bahwa aplikasi yang ada dapat mendapatkan keuntungan dari migrasi VM, harus dipastikan bahwa manfaat kinerja karena adanya relokasi yang melebihi biaya migrasi. Hal tersebut dapat dicapai dengan menghitung biaya migrasi (dalam waktu) dari VM yang satu ke *server* yang lain, dan

mmebandingkannya dengan penghematan biaya komunikasi yang akan dicapai karena proses relokasi dapat mengurangi *runtime* aplikasi.
Biaya migrasi VM pada A biasanya tergantung pada ukuran memori serta *link bandwidth* dimana proses migrasi harus terjadi. Dengan demikian, untuk VM dan target yang diberikan *server* fisik, dapat dihitung perkiraan biaya migrasi untuk ukuran memori VM dan berbanding terbalik dengan *link bandwidth* yang menghubungkan *host server* fisik untuk *server* tujuan. Dengan menggunakan perkiraan ini, kita dapat menentukan ambang batas migrasi: *volume* trafik VM bertukar dengan *server* tujuan dalam tiap pengukuran sehingga didapatkan keuntungan dari relokasi. Pada saat yang sama, migrasi lebih dari WAN tanpa *shared storage* mungkin memerlukan salinan dari penyimpanan lokal (Garey, 1979), sehingga biaya migrasi yang jauh lebih tinggi dan migrasi jauh lebih besar dari ambang batas (sebanding dengan ukuran *storage*).

BAB 3 METODOLOGI

Pada bab ini akan dijelaskan langkah-langkah yang dilakukan dalam perancangan, implementasi, analisis dan pengujian dari sistem yang dibuat. Kesimpulan dan saran disertakan sebagai catatan atas sistem dan kemungkinan arah pengembangan sistem selanjutnya. Berikut diagram alir dari pelaksanaan penelitian secara keseluruhan.



Gambar 3.1 Diagram Alir Penelitian

3.1 Studi Literatur

Studi literatur menjelaskan tinjauan teori yang digunakan untuk menunjang penelitian ini. Teori-teori tersebut adalah sebagai berikut

a. Penelitian Terkait
b. Virtualisasi
c. Pendekatan Virtualisasi
d. Hypervisor
e. *Kernel-Based Mesin virtual*
f. *Live migration*
g. *Distributed Bartering Algorithm*

3.2 Analisis Kebutuhan

Tahap analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan yang diperlukan dalam pengimplementasian *distributed bartering algorithm* pada *live migration.* Analisis kebutuhan dilakukan dengan mengindentifikasi semua kebutuhan sistem, seperti perangkat keras, perangkat lunak, kebutuhan pendukung lainnya. Analisis juga dilakukan untuk mengetahui kondisi lapangan yang ada dan menjadi dasar dari pelaksanaan perancangan dan implementasi sistem yang akan dilanjutkan pada tahap selanjutnya.

3.2.1. Kebutuhan Perangkat Keras

Penelitian yang akan membangun sistem *live migration* ini membutuhkan dua HVM (*server* fisik virtualisasi) yang mempunyai spesifikasi yang sama. Selain itu juga memerlukan beberapa perangkat jaringan. Berikut rincian dari perangkat keras yang digunakan.

a. CPU : AMD A8-6410 APU with AMD Radeon R5 Graphics, 2.00 GHz
b. RAM : 4GB
c. Hardisk : 500 GB
d. *Switch*
e. *Router*
f. Kartu jaringan
g. Media Penghubung (Kabel UTP)

3.2.2. Kebutuhan Perangkat Lunak

Perangkat lunak yang diperlukan untuk membangun sistem ini merupakan perangkat lunak *free* dan *open source*. Berikut perangkat lunak yang digunakan.

a. Sistem Operasi : Linux Fedora 14. 64 bit, *Openfiler*-2.3-x86_64 dan dsl-4.4.10 (*Damn Small Linux*)
b. *VM Hypervisor* : KVM (*Kernel-based Mesin virtual*)
c. *File sharing* : *Network File System*
d. Program : *Bash shell*

3.3 Perancangan Sistem

Perancangan implementasi *distributed bartering algorithm* pada *live migration* ini dilakukan setelah tahap analisis kebutuhan selesai dilakukan. Tahap ini diperlukan agar peneliti lebih mudah dan tepat dalam melakukan proses implementasi pada tahap selanjutnya. Dalam penelitian ini, perancangan sistem terbagi atas dua bagian yaitu perancangan jaringan dan perancangan perangkat lunak.

3.3.1. Perancangan Jaringan

Perancangan jaringan merupakan rancangan bagaimana membangun komunikasi antar perangkat keras yang ada pada sistem *live migration.* Dalam tahap ini, perangkat keras seperti *Switch*, *Router*, kartu jaringan, media penghubung (kabel UTP) tersebut dirancang agar komponen perangkat keras dapat berkomunikasi dalam sebuah segmen jaringan lokal yang sama.

3.3.2. Perancangan Perangkat Lunak

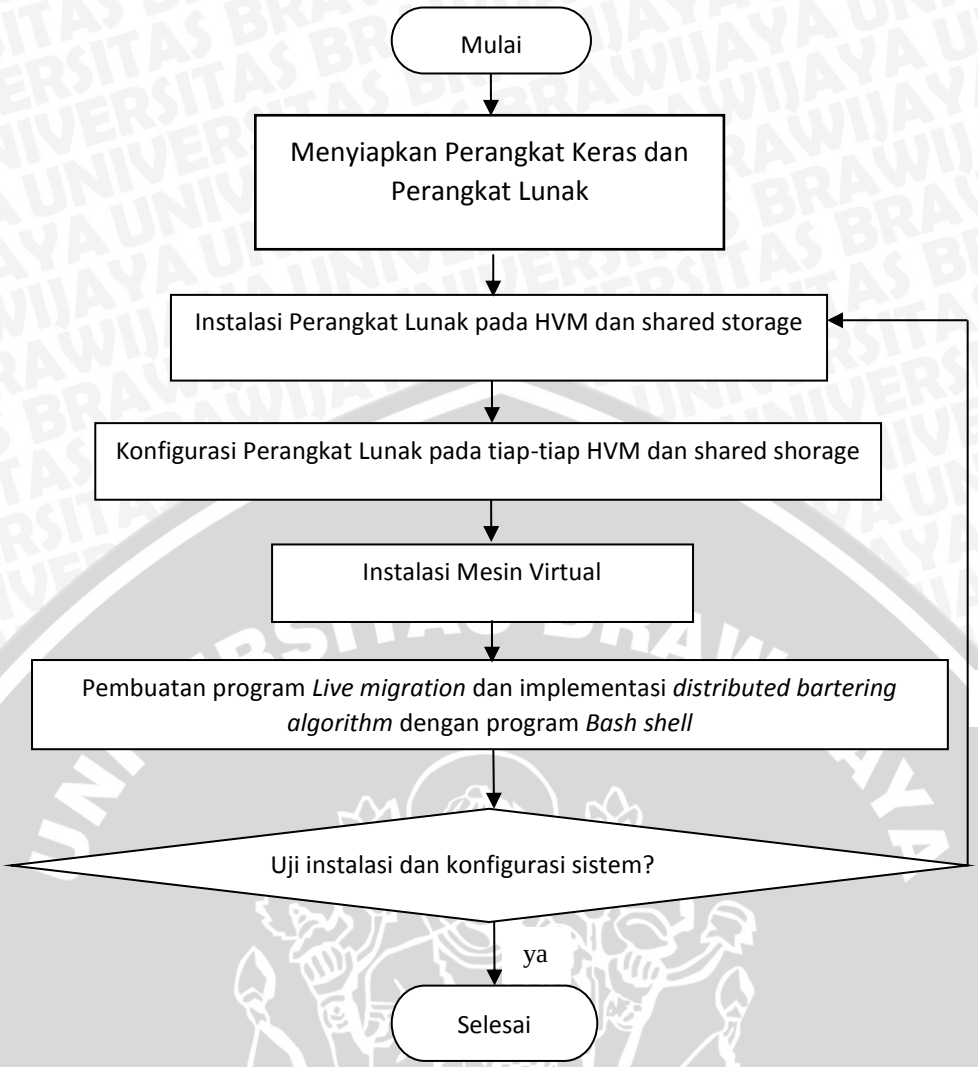
Penelitian ini memerlukan beberapa infrastruktur tambahan yang berupa perangkat lunak untuk membangun sistem implementasi *distributed bartering algorithm* yang telah dijelaskan di tahap analisis kebutuhan sebelumnya. Perangkat lunak tersebut digunakan untuk membangun dan menguji sistem pada penelitian yang dilakukan. Selain menggunakan perangkat lunak yang telah tersedia secara umum, pada penelitan ini penulis juga akan merancang program tambahan dengan bahasa *Bash shell*. Program tambahan tersebut diperlukan untuk menambahkan algoritma yang akan digunakan. Program ini berfungsi sebagai pengatur semua proses *live migration* pada sistem yang akan dibangun.

3.4 Implementasi

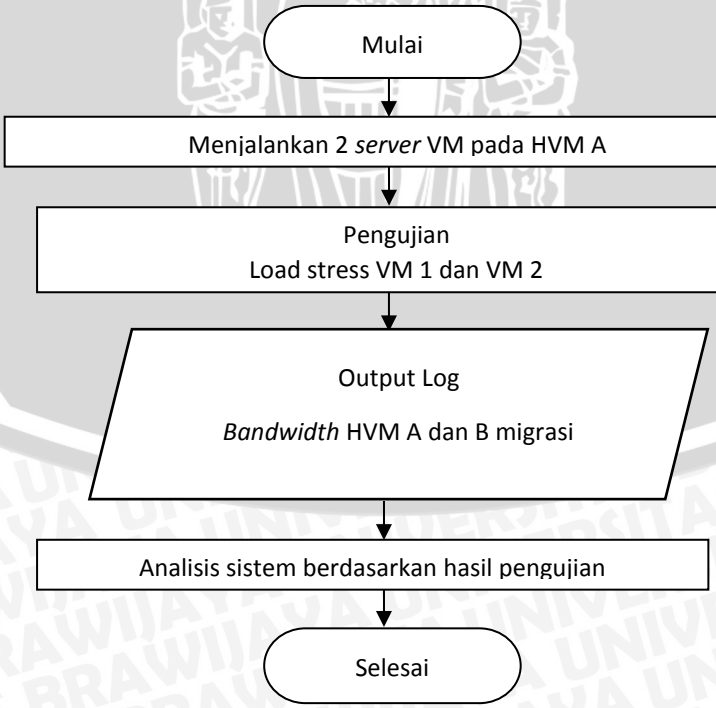
Implementasi dilakukan berdasarkan perancangan yang telah ditentukan. Tahap implementasi dapat dilihat pada gambar 3.2 dibawah ini.

3.5 Pengujian dan Analisis Sistem

Pengujian merupakan tahap yang dilakukan setelah implementasi sistem dilakukan. Tujuan dari pengujian sistem adalah untuk memastikan implementasi *distributed bartering algorithm* pada *live migration* yang dibangun dapat berjalan dengan algoritma yang digunakan dan pengujian fokus pada pengoptimalan untuk meminimalkan biaya *overhead* komunikasi jaringan antar VM yang berbeda. Hasil dari analisis tersebut akan digunakan untuk mengambil kesimpulan dan saran. Pengujian tersebut dilakukan sesuai dengan diagram alir Gambar 3.3 dibawah ini:



Gambar 3.1 Diagram Alir Implementasi



Gambar 3.2 Diagram Alir Pengujian

3.6 Pengambilan Kesimpulan

Pengambilan kesimpulan dilakukan setelah semua tahapan baik perancangan, implementasi, pengujian dan analisis sistem telah selesai dilakukan. Kesimpulan disusun berdasarkan hasil pengujian dan analisis terhadap sistem yang telah dibangun. Isi dari kesimpulan diharapkan dapat menjadi acuan untuk pengembangan dan penyempurnaan sistem dan juga dapat dijadikan sebagai tolak ukur atau referensi bagi kegiatan penelitan dengan tema yang hampir sama dengan kegiatan penelitian ini.

BAB 4 PERANCANGAN DAN IMPLEMENTASI

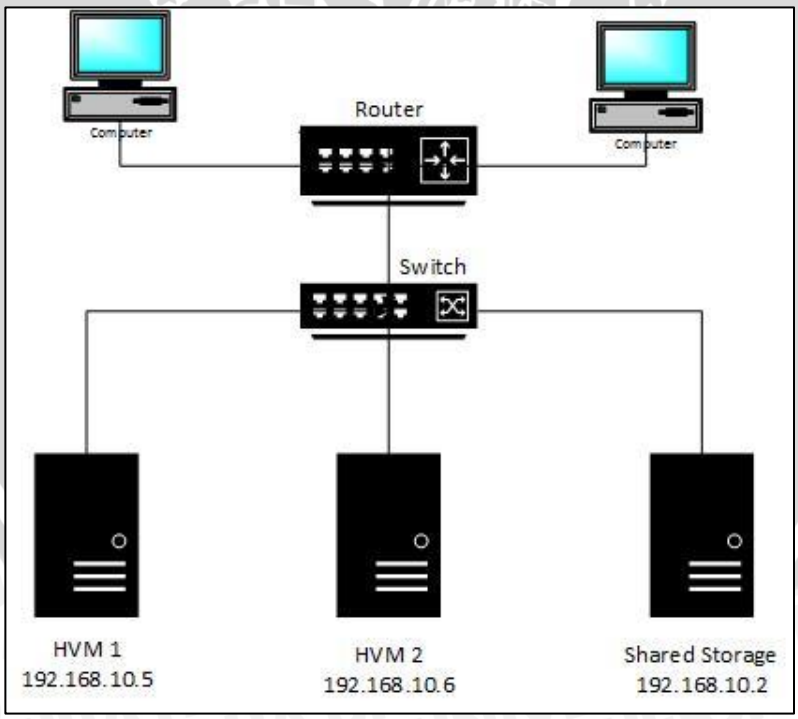
Bab ini akan membahas mengenai perancangan dari sistem yang akan dibuat. Dari perancangan sistem tersebut maka dapat dibuat skenario implementasi sistem dari penelitian ini.

4.1 Perancangan

Pada tahap perancangan ini akan membahas tentang perancangan sistem yang akan dibuat dan dibagi menjadi beberapa tahap yaitu:

4.1.1 Analisis Kebutuhan Perangkat Keras dan Lunak

Setelah memperoleh seluruh kebutuhan perangkat keras dan perangkat lunak melalui tahap analisis kebutuhan, sistem ini kemudian dirancang sesuai dengan tujuan penelitian. Perancangan perangkat keras ini terdiri atas instalasi komputer HVM, komputer *shared storage*, media penghubung (jaringan), dan perangkat keras lainnya serta dibangun pada sebuah segmen jaringan lokal yang sama. Proses perancangan dilakukan dengan menghubungkan seluruh perangkat keras dalam jaringan lokal dengan kabel UTP dan memberikan alamat jaringan ke setiap HVM dan *shared storage*. Pada penelitian ini menggunakan alamat jaringan 192.168.10.0 dan subnet mask 255.255.255.0 untuk setiap HVM dan *shared storage,* seperti yang terlihat pada Gambar 4.1 berikut ini.



Gambar 4.1 Perancangan Jaringan Sistem

4.1.2 Perancangan Perangkat Lunak

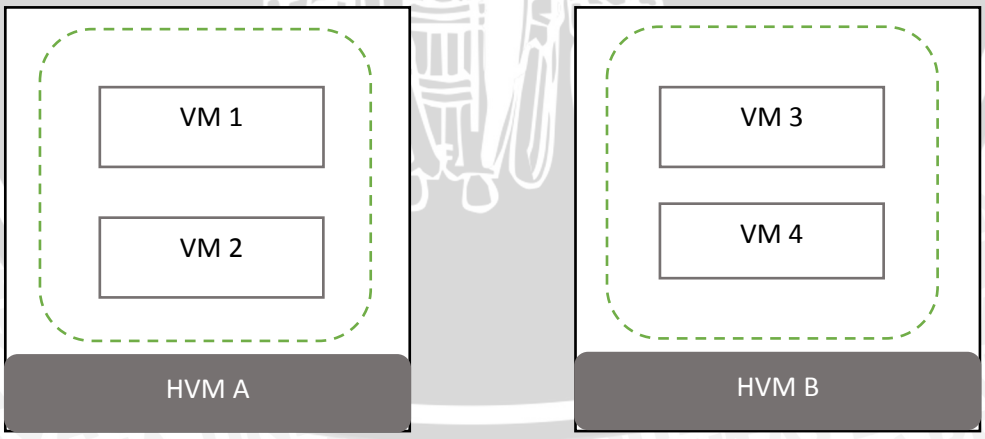
Pada penelitian ini, penulis memasang empat buah mesin virtual. Adapun alokasi sumber daya komputasi pada masing-masing *server* fisik dan mesin virtual adalah sebagai berikut:

Tabel 4.1 Alokasi Sumber Daya Komputasi

Server	CPU/vCPU	Memory	HDD
HVM A	1	2 GB	100 GB
HVM B	1	2 GB	100 GB
VM 1	1	1 GB	50 GB
VM 2	1	1 GB	50 GB
VM 3	1	1 GB	50 GB
VM 4	1	1 GB	50 GB

Tahap selanjutnya adalah dengan merancang perangkat lunak yang akan digunakan supaya dapat terbangun sebuah sistem. Berikut merupakan ilustrasi secara umum cara kerja sistem:

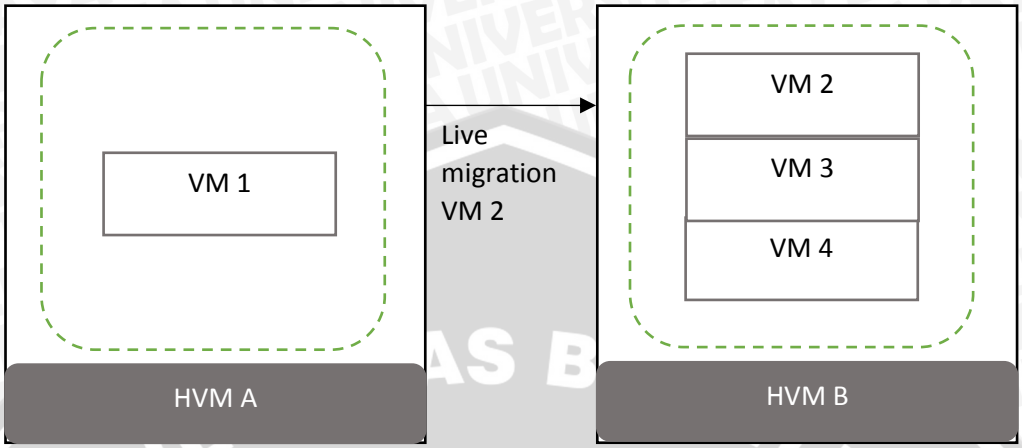
- Pada gambar dibawah ini dijelaskan bahwa sistem ketika HVM A dan HVM B dalam keadaan trafik jaringan yang stabil, sehingga baik HVM A dan HVM B tidak ada yang kelebihan beban maka tidak ada kegiatan yang dilakukan.



Gambar 4.2 Trafik jaringan dalam keadaan stabil

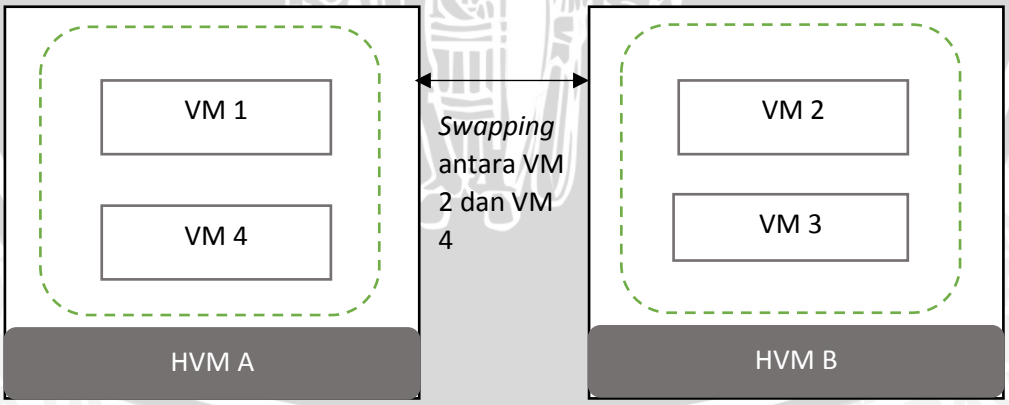
- Selanjutnya, berdasarkan parameter trafik jaringan yang tercatat pada *log* sistem maka dapat diperoleh VM yang dikategorikan memiliki beban trafik tinggi. Dalam penelitian ini, penulis menentukan ambang batas beban trafik adalah 400 Mbits/sec. Dari hasil perolehan tersebut maka VM yang melebihi

ambang batas/kelebihan beban akan dipindahkan atau dialokasikan pada HVM yang lebih proporsional. Berikut merupakan gambaran secara umum proses perpindahan VM dari HVM A ke HVM B.



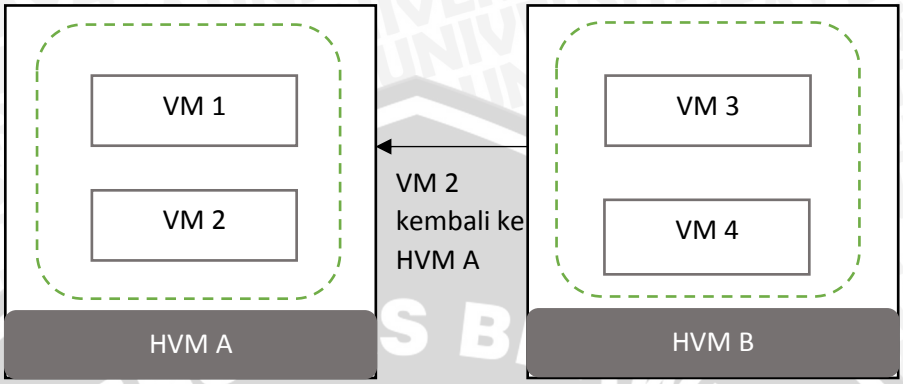
Gambar 4.3 Proses Migrasi VM

- Dalam penelitian ini ditentukan juga untuk memulai proses *swapping* maka ambang batas beban trafik adalah sekitar 200-300 Mbits/sec. Proses *swapping* merupakan salah satu bentuk pengimplementasian dari *distributed bartering algorithm.* Pada Gambar 4.3 dijelaskan bahwa proses *swapping* terjadi ketika HVM A memiliki trafik jaringan yang tinggi, jumlah beban dari VM 1 dan VM 2 melewati ambang batas, yang sudah ditentukan sedangkan HVM B memiliki trafik yang stabil sehingga terjadi proses komunikasi antara VM dari HVM A yang memiliki *throughput* tinggi dengan VM dari HVM B yang memiliki *throughput* sesuai dengan batas yang ditentukan.



Gambar 4.4 Proses *Swapping* VM

- Selanjutnya ketika trafik jaringan HVM A dan HVM B sama-sama tinggi maka tidak ada proses migrasi maupun *swap* yang dilakukan. Hal ini mengingat karena jika kedua HVM memiliki trafik yang tinggi maka tidak ada hal yang bisa dilakukan.



Gambar 4.5 Proses *Remigration* VM

4.1.3 Penggunaan Perangkat Lunak

Adapun beberapa perangkat lunak yang digunakan untuk membangun sistem prediktif *live migration*, antara lain:

1. Sistem Operasi

Sistem operasi yang digunakan dalam penelitian ini adalah sistem operasi Fedora-14-x86_64, *Openfiler*-2.3-x86_64 dan dsl-4.4.10 (*Damn Small Linux*). Fedora-14-x86_64 digunakan sebagai sistem operasi di komputer HVM sedangkan *Openfiler*-2.3-x86_64 digunakan di komputer *shared storage.* Sistem operasi tersebut dipilih karena pada penelitian-penelitian sebelumnya Fedora dapat berjalan dengan baik dengan *KVM Hypervisor* yang digunakan pada penelitian ini. *Openfiler* dipilih karena sistem operasi ini secara khusus digunakan sebagai *shared storage*, sangat mudah digunakan dan dipahami serta memiliki *user interface* berbasis web. Kemudian penulis memilih *Damn Small Linux (DSL),* dikarenakan DSL merupakan jenis Linux yang sangat ringan dan alokasi *storage* yang dibutuhkan sangat kecil, mengingat karena adanya keterbatasan sumber daya komputasi pada penelitian ini.

2. Hypervisor

Hypervisor merupakan perangkat lunak yang berfungsi untuk menjalankan dan membuat mesin virtual yang dipasang pada setiap komputer HVM. Pada penelitian ini *KVM (Kernel Virtual Machine*) dipilih sebagai *VM Hypervisor*.

3. File sharing

Jenis *File sharing* yang digunakan pada penelitian ini adalah *Network File System (NFS).* NFS berfungsi untuk mengakses *virtual disk* yang tersimpan pada komputer *shared storage*, sehingga berfungsi untuk mengurangi besarnya data

VM yang harus dipindahkan sehingga pada saat proses migrasi dilakukan, waktu yang diperlukan dapat lebih cepat sehingga lebih efektif.

4. ***System Monitoring***

   System Monitoring perlu dipasang di tiap komputer HVM karena perangkat lunak tersebut digunakan untuk mendapatkan informasi presentase beban kerja yang ditanggung oleh *server* fisik maupun mesin virtual. Perangkat lunak tambahan itu adalah *sysstat* (monitoring beban kerja HVM) dan *virt-top* (monitoring beban kerja seluruh mesin virtual). Kedua perangkat lunak tersebut digunakan pada masing-masing HVM.

5. ***Bash shell***

   Digunakan untuk membangun sistem pada penelitian ini dengan metode *distributed bartering algortihm* diperlukan beberapa program berkas *Bash shell. Bash shell* merupakan alat yang digunakan untuk berinteraksi dengan sistem operasi komputer. Saat berkas program dijalankan, *shell* bertindak untuk mengambil perintah yang dimasukan, menentukan program mana dan apa yang perlu dijalankan, lalu menampilkannya. Program *Bash shell* tersebut berisi beberapa perintah, seperti monitoring, seleksi kondisi, perpindahan mesin virtual, serta pencatatan *log* dari sistem.

4.1.4 Algoritma Program

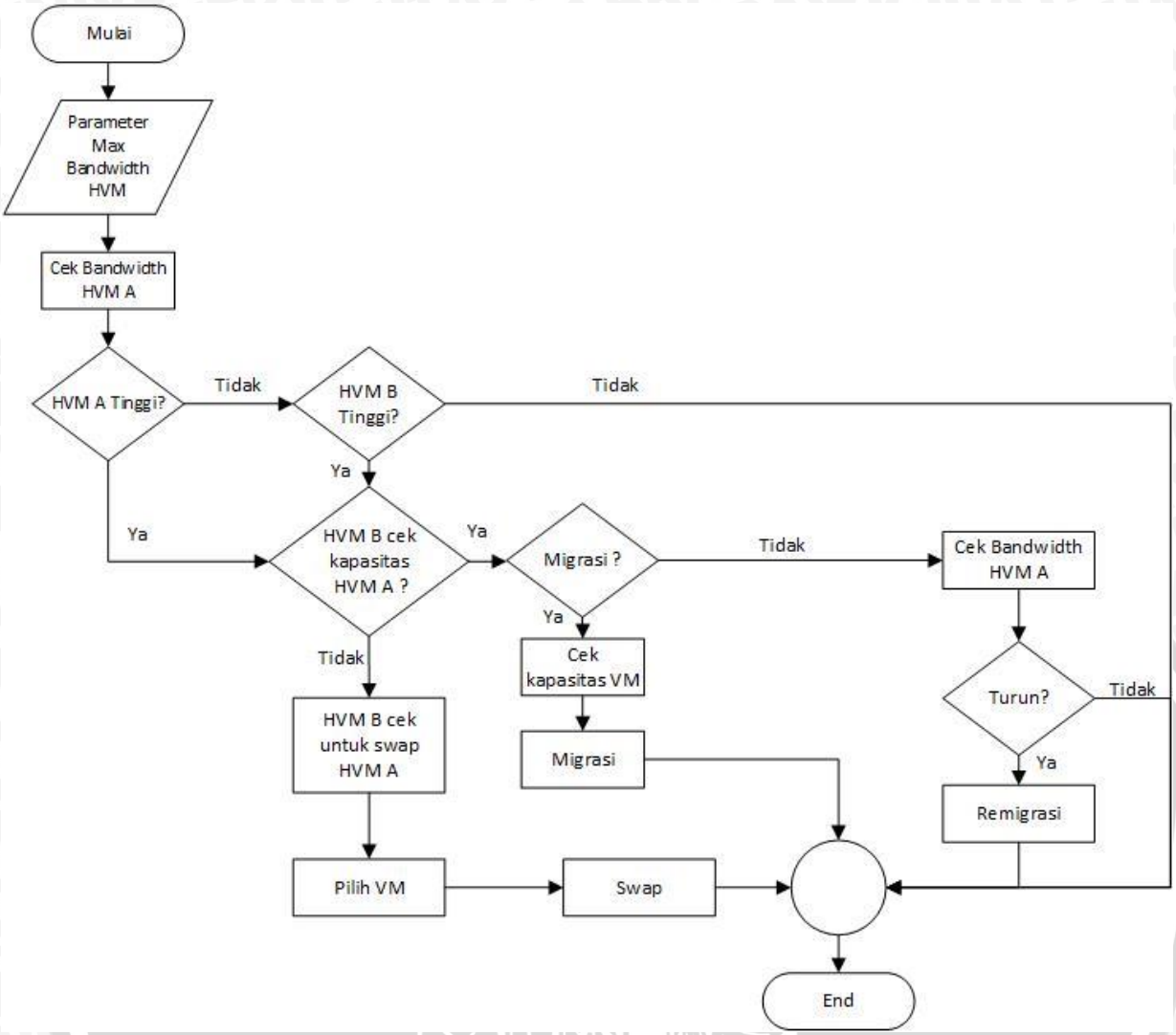
Untuk dapat melakukan proses implementasi *distributed bartering algortihm* memerlukan sebuah program tambahan *Bash shell* yang dijalankan secara berkala dan terpusat pada HVM A (agen barter A). Adapun diagram alir algoritma program tambahan tersebut, pada Gambar 4.6 dibawah.

4.2 Implementasi

4.2.1 Instalasi HVM

Tahap awal yang perlu dilakukan dalam membangun sistem yang akan dibuat adalah dengan menghubungkan komponen-komponen yang diperlukan sesuai dengan perancangan yang telah dibuat sebelumnya.

Tahap berikutnya adalah proses instalasi perangkat lunak yang diperlukan pada HVM satu dan HVM dua. Instalasi yang pertama kali dilakukan adalah instalasi sistem operasi Fedora-14-x86_64. Selanjutnya, proses instalasi perangkat lunak lainnya, seperti KVM *Hypervisor*, Qemu, NFS, dan *System Monitoring* (*sysstat* dan *virt-top*).



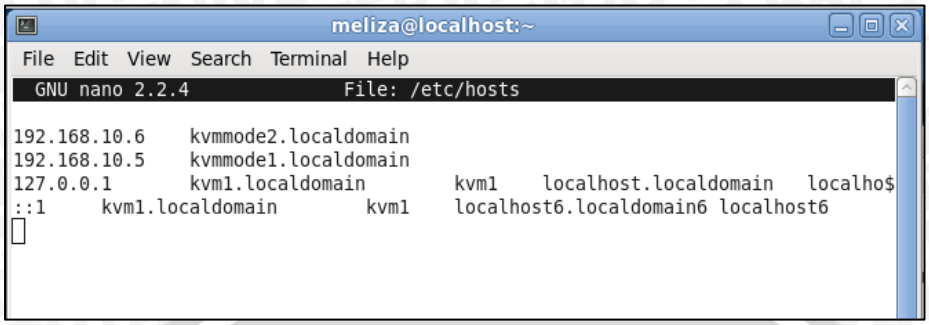
Gambar 4.6 Algoritma Sistem

4.2.2 Konfigurasi Perangkat Lunak

Setelah melakukan proses instalasi perangkat lunak yang dibutuhkan sistem, langkah selanjutnya adalah mengkonfigurasikan beberapa sistem keamanan dan sistem operasi. Langkah pertama yang dilakukan adalah dengan mematikan *firewall* atau dengan membuka beberapa *port* saja yang hanya diperlukan untuk seluruh HVM dan komputer *shared storage.* Hal ini dilakukan supaya HVM dapat mengakses data yang ada pada *shared storage* dan dapat menjalankan proses *live migration* mesin virtual. Berikutnya, membuat *RSA key public* sehingga proses komunikasi antar HVM tidak dibatasi oleh aturan authentifikasi sistem operasi dan dalam proses pemindahan mesin virtual dari HVM A ke HVM B atau sebaliknya tanpa perlu memasukan *password*.

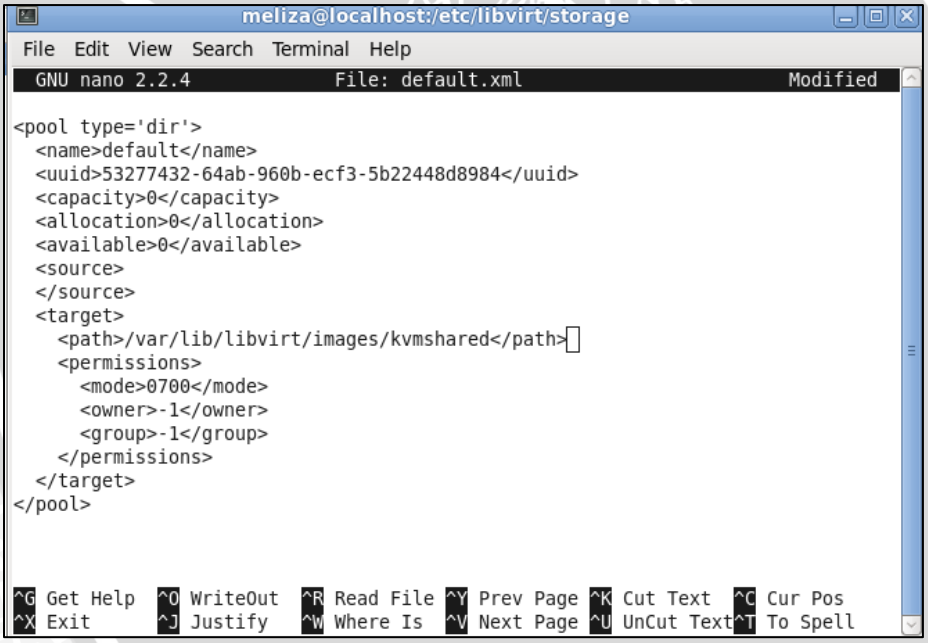
Langkah kedua yaitu dengan melakukan konfigurasi terhadap beberapa berkas yang berkaitan dengan penggunaan NFS dan KVM. Untuk pengunaan NFS, diperlukan konfigurasi berkas *hosts* pada direktori */etc* untuk menerjemahkan

nama HVM atau *hostname* ke dalam alamat IP, seperti yang terlihat pada Gambar 4.7.



Gambar 4.7 Konfigrasi /etc/*hosts*

Sedangkan untuk KVM, diperlukan konfigurasi berkas *default.xml* yang terdapat pada direktori */etc/libvirt/storage/* yang didalamnya berisi pool *storage KVM*. Hal ini dilakukan dengan mengubah *path* pool *storage* agar berkas NFS disimpan pada direktori yang telah dibuat yaitu pada direktori */var/lib/libvirt/images/kvmshared*. Berikut hasil konfigurasi default.xml.

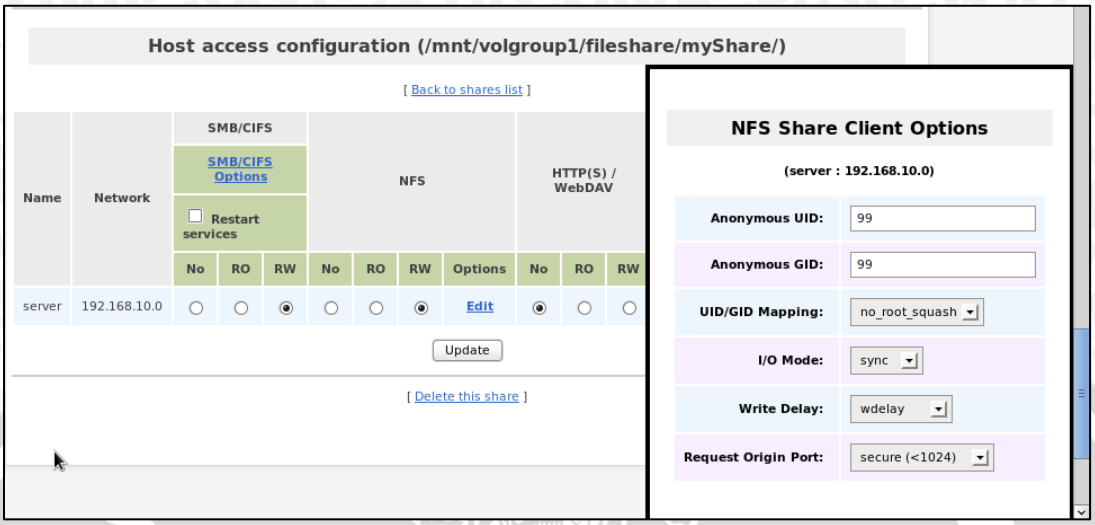


Gambar 4.8 Konfigrasi /etc/libvirt/storage/

4.2.3 Implementasi *Shared storage*

Proses implementasi *shared storage* diawali dengan proses instalasi sistem operasi yaitu sistem operasi *Openfiler*-2.3-x86_64. *Openfiler* merupakan sebuah sistem operasi pengelola penyimpanan data. *Openfiler* dapat digunakan sebagai basis untuk membangun *Network Attached Storage* (NAS) dan perangkat aplian *Storage Area Network* (SAN). Sistem operasi ini menggabungkan teknologi *open source* menjadi solusi yang kompak dengan *frontend* dan GUI berbasis web yang mudah diatur. Pada *Openfiler* sudah tersedia NFS dan beberapa fitur lainnya yang dapat diaktifkan maupun dinonaktifkan dengan mudah.

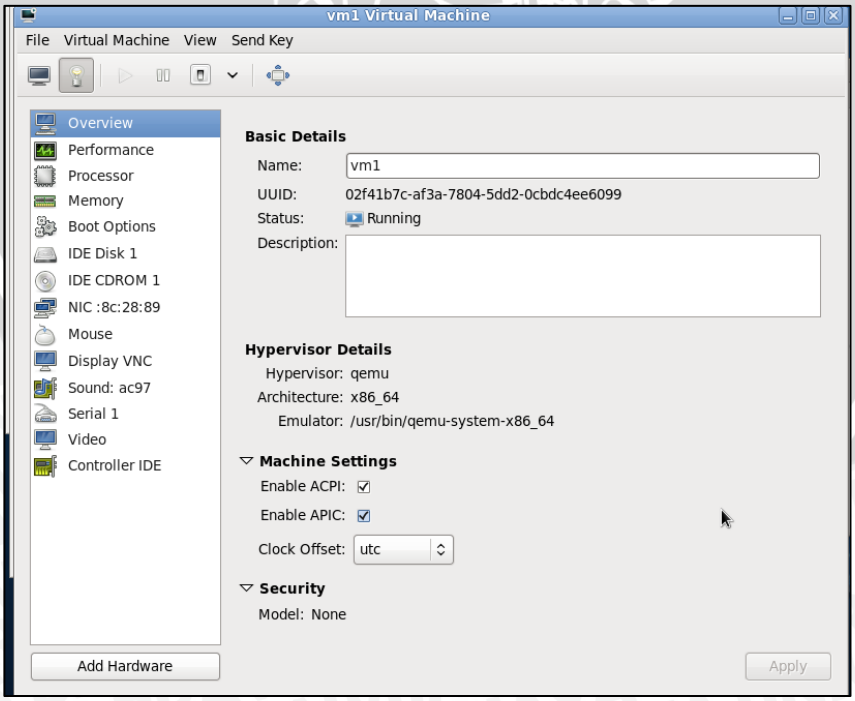
Tahap selanjutnya, diperlukan konfigurasi berkas NFS untuk memberikan hak akses baca tulis untuk segmen jaringan tertentu, dalam penelitian ini berada pada segmen jaringan 192.168.10.0/24. Setelah itu, proses mount direktori NFS di seluruh HVM agar mengakses data pada *shared storage.* Berikut konfigurasi NFS pada *shared storage*.



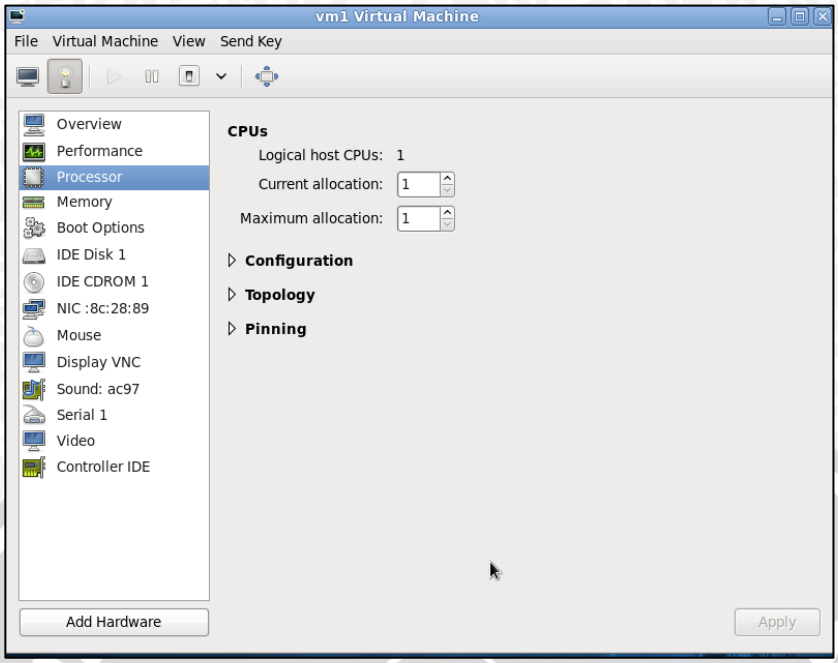
Gambar 4.9 Konfigurasi NFS pada *shared storage*

4.2.4 Implementasi Mesin Virtual

Tahapan implementasi ini diawali dengan proses instalasi mesin virtual melalui perangkat lunak *KVM Hypervisor* dimana alokasi untuk sumber daya komputasi sesuai dengan rancangan yang telah dilakukan sebelumnya. Adapun spesifikasi mesin virtual yang telah diinstal dapat pada gambar 4.10.

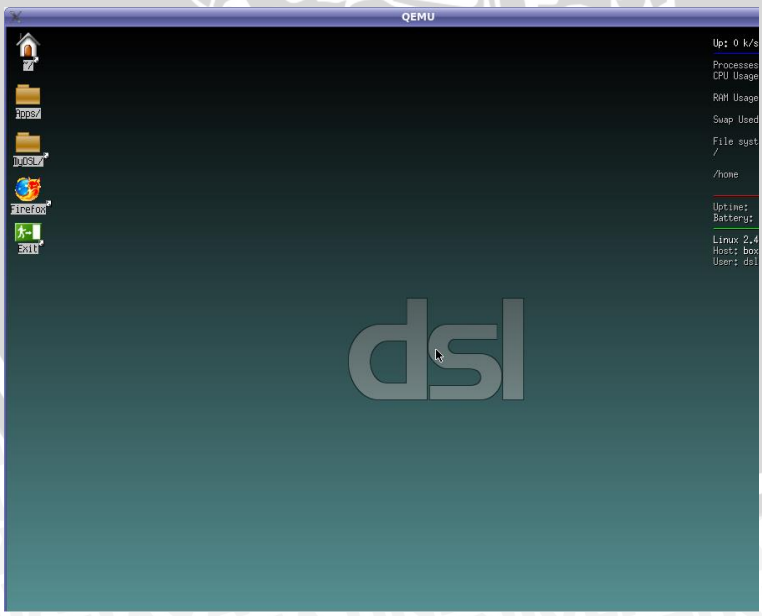


Gambar 4.10 Detail Umum Spesifikasi Mesin Virtual 1



Gambar 4.11 Detail Processor Spesifikasi Mesin Virtual 1

Mengingat alokasi yang diberikan tidak besar maka diperlukan sistem operasi yang simple, dan tidak berat. Maka sistem operasi yang tepat dipasang pada mesin virtual tersebut adalah sistem operasi *Damn Small Linux* seperti pada Gambar 4.12. Sedangkan untuk jaringan, mesin virtual tidak memerlukan konfigurasi khusus karena IP dapat diperoleh secara otomatis dengan jaringan *bridge*.



Gambar 4.12 Damn Small Linux

Implementasi yang dilakukan selanjutnya adalah pembuatan program tambahan yang berupa berkas *Bash shell* dan teks. Berkas-berkas tersebut berisi perintah-perintah *shell* yang digunakan untuk membangun sistem pada penelitian

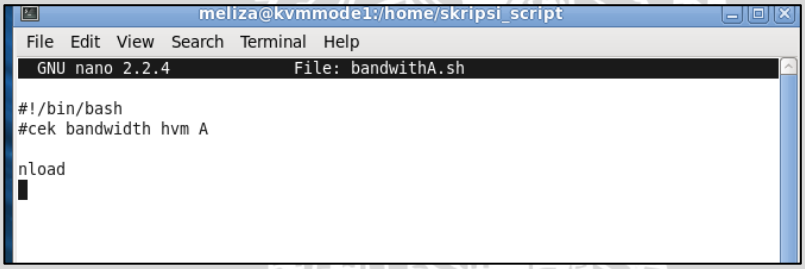
ini, dimana tiap-tiap berkas memiliki fungsi tersendiri. Berkas tersebut disimpan pada direktori */home/skripsi_script* HVM A yang merupakan HVM utama. Berikut daftar berkas-berkas untuk program tambahan yang dibuat penulis.

- ***bandwidthA.sh*** : berkas *shell* ini berisi perintah untuk memonitoring *bandwidth* yang berjalan pada HVM A. Proses monitoring dilakukan dengan menggunakan *tool* nload.
- ***bandwidthA.txt:*** berisi hasil dari perintah yang dijalankan pada *bandwidthA.sh*
- ***main.sh*** : merupakan berkas *shell* utama yang mengatur semua proses yang ada pada sistem penelitian ini.
- ***totalvm.sh*** **:** berkas *shell* yang berisi perintah *virsh list* yaitu untuk mencari tahu berapa jumlah mesin virtual yang berjalan di HVM A.
- ***totalvm_run.txt*** : berkas ini berisi jumlah mesin virtual yang berjalan pada HVM A.

Berdasarkan daftar berkas program tambahan untuk sistem penelitian diatas, maka terdapat empat berkas *Bash shell* yaitu *hvm1.sh, totalvm.sh* dan *main.sh.* Adapun penjelasan dari masing-masing berkas tersebut, sebagai berikut.

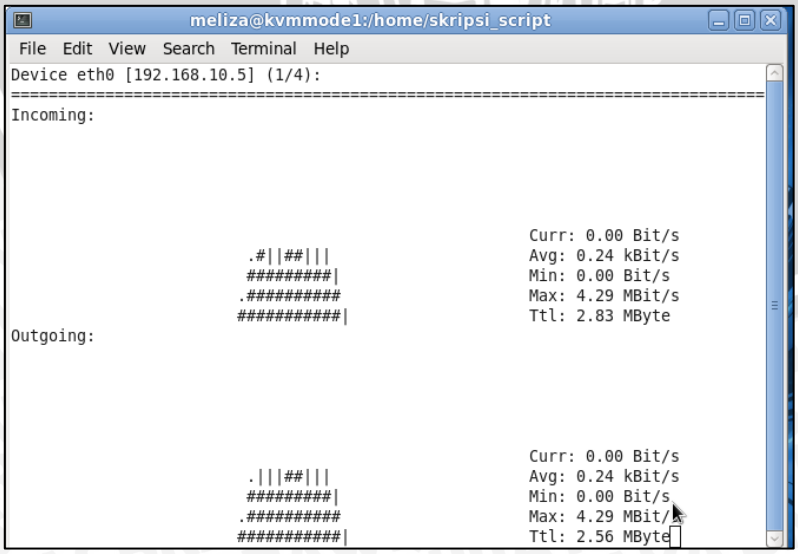
- ***bandwidth*A.sh**

Berkas *shell bandwidthA.sh* ini dibangun untuk menjalankan perintah monitoring hasil *bandwidth* pada HVM A. Perintah monitoring tersebut dapat dilihat pada Gambar 4.13.



Gambar 4.13 Hasil *bandwidth*A.sh

- ***bandwidth*A.txt**



Gambar *4.14* Hasil bandwidthA.txt

- **main.sh**

Berkas ini merupakan berkas utama dari sistem penelitian yang dibuat dimana berkas ini semua proses yang berjalan. Isi dari main.sh adalah perintah parameter *variable*, pengambilan data, klasifikasi mesin virtual, migrasi atau remigrasi, proses *swapping* dan pencatatan log sistem dapat dilihat pada Lampiran 1. Berikut merupakan beberapa contoh dari perintah-perintah yang ada pada main.sh

- Memasukkan *variable*

```
#!/bin/bash
maxhvmA=400
maxhvmB=400
tgl=`date`
migration=`cat /home/skripsi_script/migration.txt | head -
n 1`
```

- Pengambilan data

```
bandwidthA=`cat /home/skripsi_script/bandwidhtA.txt | head
-n 1`
bandwidthB=`cat /home/skripsi_script/bandwidhtB.txt | head
-n 1`
```

- Proses migrasi atau remigrasi

```
if [ $bandwidth-vm_1+$bandwidth-vm_2 -gt $maxhvmA ]
then
   totalvm=`virsh list | awk 'END {print NR-2}'`
     if [ $totalvm -eq 2 ];then
         if [ $bandwidth-vm_1 -lt $maxhvmB - ($bandwidth-
vm_3+$bandwidth-vm_3)] then;
             elif [ $bandwidth-vm_2 -lt $maxhvmB -
($bandwidth-vm_3+$bandwidth-vm_3)] then;
             vmtarget=1
             else
             vmtarget=2
fi
```

- Proses *swapping*

```
elif
    if ($bandwidth-vm_1 + $bandwidth-vm_4 -lt $maxhvmB &&
    $bandwidth-vm_2 + $bandwidth-vm_3 -lt $maxhvmBA) {
    //cek SWAP VM1 dg VM3
         then ;
         virsh migrate --live vm_3
    qemu+ssh://kvmmode2.localdomain/system
         virsh --connect
    qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \
    migrate --live vm_$vmmigrate qemu+ssh:///system

    elif ($bandwidth-vm_1 + $bandwidth-vm_3 -lt $maxhvmB
    && $bandwidth-vm_2 + $bandwidth-vm_4 < $maxhvmA) {
    //cek SWAP VM1 dg VM4
         then;
         virsh migrate --live vm_1
    qemu+ssh://kvmmode2.localdomain/system
         virsh --connect
    qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \
    migrate --live vm_$vmmigrate qemu+ssh:///system

    elif ($bandwidth-vm_2 + $bandwidth-vm_4 -lt $maxhvmB
    && $bandwidth-vm_1 + $bandwidth-vm_3 -lt $maxhvmA) {
    //cek SWAP VM2 dg VM3
         then;
         #$vm2 = "(swap dg VM3 ke B)";
         #$vm3 = "(swap dg VM2 ke A)";
         virsh migrate --live vm_2
    qemu+ssh://kvmmode2.localdomain/system
         virsh --connect
    qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \
    migrate --live vm_$vmmigrate qemu+ssh:///system
    elif ($bandwidth-vm_2 + $bandwidth-vm_3 -lt $maxhvmB
    && $bandwidth-vm_1 + $bandwidth-vm_4 -lt $maxhvmA) {
    //cek SWAP VM2 dg VM4
         #$vm2 = "(swap dg VM4 ke B)";
         #$vm4 = "(swap dg VM2 ke A)";
         virsh migrate --live vm_4
    qemu+ssh://kvmmode2.localdomain/system
```

```
        virsh --connect
    qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \
    migrate --live vm_2 qemu+ssh:///system
    fi
fi
```

BAB 5 PENGUJIAN DAN ANALISIS

Bab ini akan membahas tahap pengujian dari implementasi sistem yang telah dibuat dan menganalisa hasil pengujian sistem tersebut. Tujuan dilakukannya pengujian ini adalah untuk mengetahui semua kebutuhan fungsional dan non-fungsional yang telah dirancang sebelumnya apakah telah terpenuhi.

Dalam proses pengujian dilakukan beberapa perubahan yang disesuaikan dengan kebutuhan pengujian pada program yang digunakan. Terdapat beberapa skenario yang disiapkan untuk pengujian yang akan dilakukan.

Pada tahap analisis hasil pengujian dilakukan dengan cara membandingkan data yang ada pada pengujian dengan hipotesis dan akan ditarik kesimpulan dari setiap pengujian yang ada.

5.1 Pengujian

Pengujian dilakukan sesuai dengan perancangan topologi yang sudah dibahas pada bab sebelumnya serta tidak memakai alat uji atau program uji tambahan tetapi cukup dengan membuat sibuk trafik jaringan dengan melakukan ping antar *server* secara terus menerus.

Pengujian dilakukan dengan menggunakan tiga skenario *stress load* yang mengacu pada diagram alir pengujian pada bab metode penelitian sebelumnya. Pada setiap skenario pengujian yang dilakukan, penulis mengambil rentan waktu selama 100 menit. Dalam rentan waktu 100 menit tersebut sistem akan bekerja secara reaktif terhadap trafik aliran data.

Selanjutnya, data hasil pengujian akan diperoleh melalui berkas log.txt yang telah dibuat oleh penulis. Migrasi adalah proses dimana terdapat mesin virtual yang berpindah dari HVM A ke HVM B, *swapping* adalah proses dimana terdapat mesin virtual yang saling bertukar antara HVM A dan HVM B sedangkan remigrasi adalah proses dimana mesin virtual berpindah kembali dari HVM B ke HVM A. Data yang diperoleh dari hasil pengujian ini nantinya dianalisis dan akan digunakan untuk menarik kesimpulan dan saran. Berikut penjelasan dari hasil pengujian skenario satu, skenario dua, dan skenario tiga serta hasil analisis yang dilakukan.

5.1.1 Skenario Satu

Pada pengujian skenario satu, *stress load* yang dilakukan hanya ditujukan pada HVM A. Pada proses ini, HVM A yang terdiri dari VM 1 dan VM 2 diberikan *stress load* sesuai dengan hasil *bandwidth* yang diperoleh dengan menggunakan *nload*.

Tujuan dari pengujian skenario satu ini dilakukan adalah untuk mengetahui proses migrasi dengan mengetahui perilaku sistem saat HVM A ketika diberikan *stress load* sesuai dengan karakteristik beban kerjanya. *Stress load* dilakukan pada VM 1 dengan menjalankan ping terhadap mesin virtual lainnya secara bersamaan dalam kurun waktu 100 menit. Hasil dari pengujian skenario satu adalah analisis

fungsional sistem serta proses migrasi ketika *bandwidth* pada HVM A terkadang sibuk dan terkadang stabil. Berikut hasil pengujian skenario satu.

Tabel 5.1 Hasil Pengujian Skenario Satu

Menit	Beban *Bandwidth* (Mbits/sec)		Perpindahan	
	HVM A	HVM B	Migrasi	Remigrasi
0	210	153	-	-
5	423	256	√	-
10	426	157	√	-
15	152	316	-	-
20	255	360	-	-
25	202	146	-	√
30	476	241	√	-
35	256	112	-	-
40	234	106	-	-
45	307	221	-	-
50	216	287	-	-
55	180	356	-	√
60	251	218	-	-
65	291	188	-	-
70	279	178	-	-
75	450	167	√	-
80	205	363	-	-
85	301	169	-	-
90	172	339	-	√
95	254	184	-	-
Rata-rata	277	225,85		

Berdasarkan hasil pengujian skenario satu yang diperlihatkan pada Tabel 5.1, dapat dilihat bahwa HVM A sebagai target pengujian memiliki trafik yang naik turun tiap beberapa menit. Pergerakan beban *troughput* pada VM 1 tersebut menimbulkan pergerakan pula pada beban trafik HVM A atau HVM B yang selaras. Sedangkan pada VM 2 tidak dilakukan pengujian *stress load*, beban *troughput* tidak mencapai angka 400 Mbits/sec. Tabel 5.1 memperlihatkan bahwa sistem melakukan pembagian beban kerja yaitu dengan melakuan migrasi VM 1 di menit ke-5, ke-10, ke-30 dan ke-75, dan di menit ke-25, ke-55 dan ke-90 melakukan remigrasi. Selain itu, dapat disimpulkan bahwa sistem mampu berjalan sesuai dengan fungsinya serta mampu menyeimbangkan beban trafik jaringan HVM A dan HVM B pada angka masing-masing 277 Mbits/sec dan 225,85 Mbits/sec.

5.1.2 Skenario Dua

Pada pengujian skenario dua, *stress load* ditujukan pada HVM A yang memiliki *stress load* yang sangat tinggi sedangkan pada HVM B memiliki *stress load*

yang berada pada rentan 200-300 Mbits/sec. Pengujian ini bertujuan untuk mengetahui perilaku sistem ketika dua buah mesin virtual berjalan dengan *stress load* yang cukup tinggi serta melihat proses pertukaran mesin (*swapping*) yang dilakukan. Hasil pengujian skenario dua ditunjukkan pada tabel 5.2.

Tabel 5.2 *Hasil Pengujian Skenario Dua*

Menit	Beban *Bandwidth* (Mbits/sec)		Perpindahan		
	HVM A	HVM B	Migrasi	Remigrasi	*Swapping*
0	169	206	-	-	-
5	223	304	-	-	-
10	305	385	-	-	-
15	268	389	-	-	-
20	189	297	-	√	-
25	269	212	-	-	-
30	422	294	-	-	VM 1 dan VM 3
35	424	259	√	-	-
40	323	276	-	-	-
45	196	214	-	-	-
50	286	304	-	-	-
55	461	122	√	-	-
60	210	278	-	-	-
65	215	274	-	-	-
70	189	141	-	√	-
75	300	315	-	-	-
80	415	200	√	-	-
85	231	317	-	-	-
90	152	368	-	√	-
95	268	282	-	-	-
Rata-rata	275,75	271,85			

Hasil pengujian skenario dua yang diperlihatkan pada Tabel 5.2, bahwa HVM A dan HVM B sebagai target pengujian memiliki beban kerja yang stabili pada kurun waktu pengujian yaitu 100 menit sedangkan pada HVM A dan HVM B berjalan sesuai dengan karakteristik beban kerjanya. Beban *bandwidth* HVM A mencapai nilai yang sangat tinggi yaitu 461 Mbits/sec pada menit ke-55 sedangkan pada HVM B menunjukan nilai yang sangat tinggi yaitu 389 Mbits/sec pada menit ke-15.

Pengujian skenario dua ini memperlihatkan bahwa proses *swapping* dapat berjalan ketika beban *bandwidth* HVM A sangat tinggi dan beban *bandwidth* HVM B berjalan sesuai dengan batas yang ditentukan yaitu sekitar 200-300 Mbits/sec.

5.1.3 Skenario Tiga

Pada pengujian skenario tiga, *stress load* ditujukan masih pada tiga buah mesin virtual yaitu VM 1, VM 2, VM 3 dan VM 4 tetapi dengan keadaan *stress load*

yang dipusatkan pada VM 3. Untuk VM 1 dan VM 2 *stress load* dilakukan sesuai dengan karakteristik beban kerjanya sedangkan VM 3, dilakukan dengan beragam *stress load*. Berikut hasil dari pengujian skenario tiga. Berdasarkan hasil pengujian skenario tiga, dapat dilihat pada Tabel 5.3 bahwa beban *bandwidth* pada HVM B lebih tinggi dari percobaan sebelumnya. Meskipun dapat dilihat bahwa beban kerja HVM A mencapai angka 426 Mbits/sec pada menit ke-45, sistem dapat melakukan penyeimbangan beban kerja dengan melakukan perpindahan mesin virtual sehingga beban kerja. Dari hasil pengujian pada skenario tiga ini, maka dapat disimpulkan bahwa, sistem mampu berjalan dengan baik yaitu dapat penyeimbangan beban kerja serta proses pemindahan mesin virtual dapat dilakukan dengan baik.

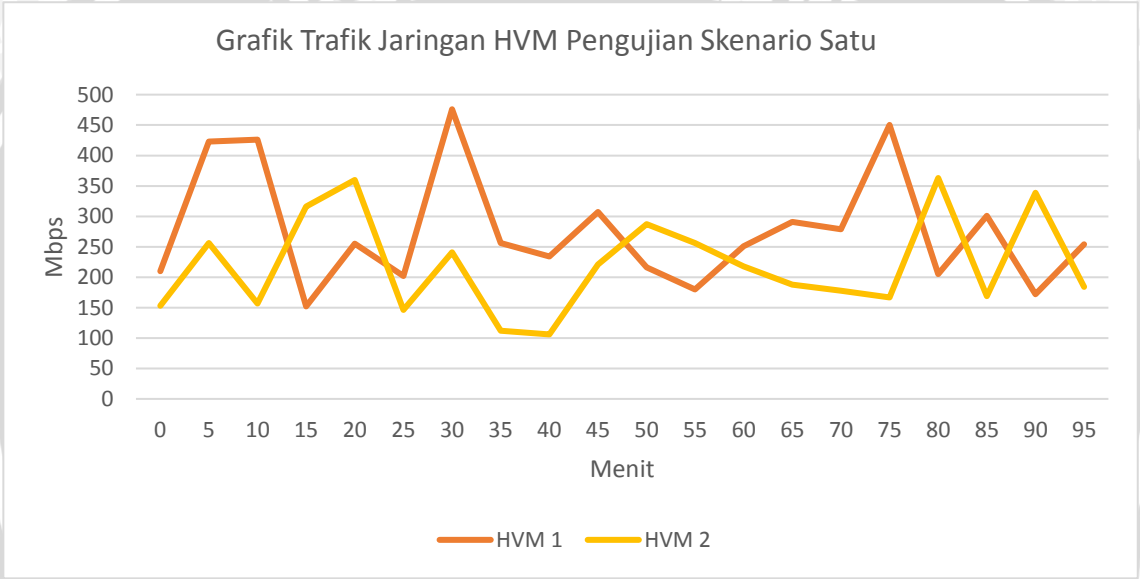
Tabel 5.3 Hasil Pengujian Skenario Tiga

Menit	Beban *Bandwidth* (Mbits/sec)		Perpindahan		
	HVM A	HVM B	Migrasi	Remigrasi	*Swapping*
0	153	344	-	-	-
5	206	376	-	-	-
10	401	267	-	-	VM_2 dan VM_4
15	325	386	-	-	-
20	347	382	-	-	-
25	245	401	-	-	-
30	253	397	-	-	-
35	415	402	-	-	-
40	402	310	-	-	-
45	416	267	-	-	VM_3 dan VM_1
50	370	365	-	-	-
55	360	380	-	-	-
60	205	371	-	-	-
65	185	350	-	-	-
70	195	400	-	-	-
75	315	396	-	-	-
80	260	382	-	-	-
85	400	239	-	-	VM_1 dan VM_3
90	350	370	-	-	-
95	352	361	-	-	-
Rata-rata	307,65	357,3			

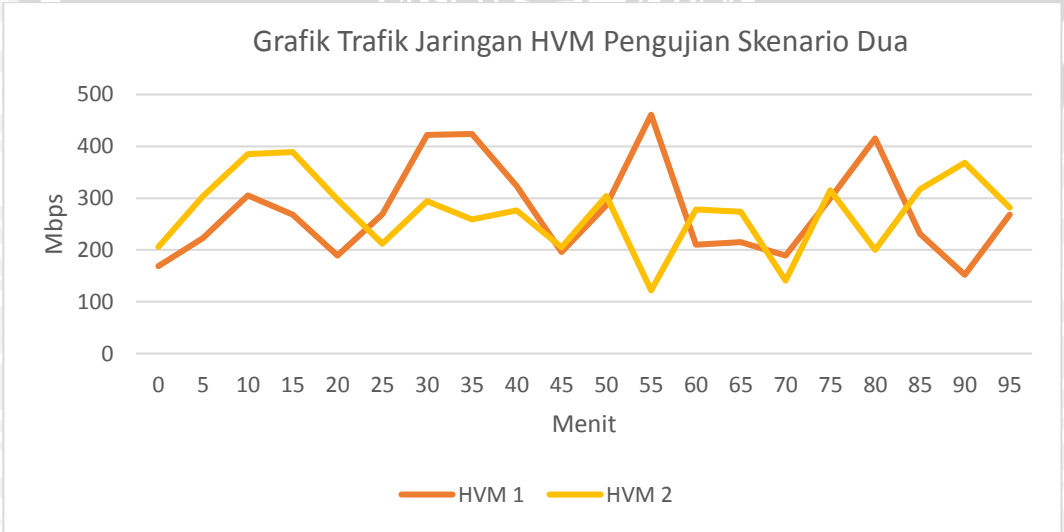
5.2 Analisis Keseluruhan Sistem

Hasil dari pengujian skenario satu, dua, dan tiga yang telah dilakukan sebelumnya menunjukan bahwa sistem yang telah dibangun dapat berjalan dengan baik secara fungsional. Sistem dapat berjalan lancar baik yaitu karena dapat melakukan *live migration* antar VM dan dapat melalukan proses *swapping* sesuai dengan ambang batas yang ditentukan.

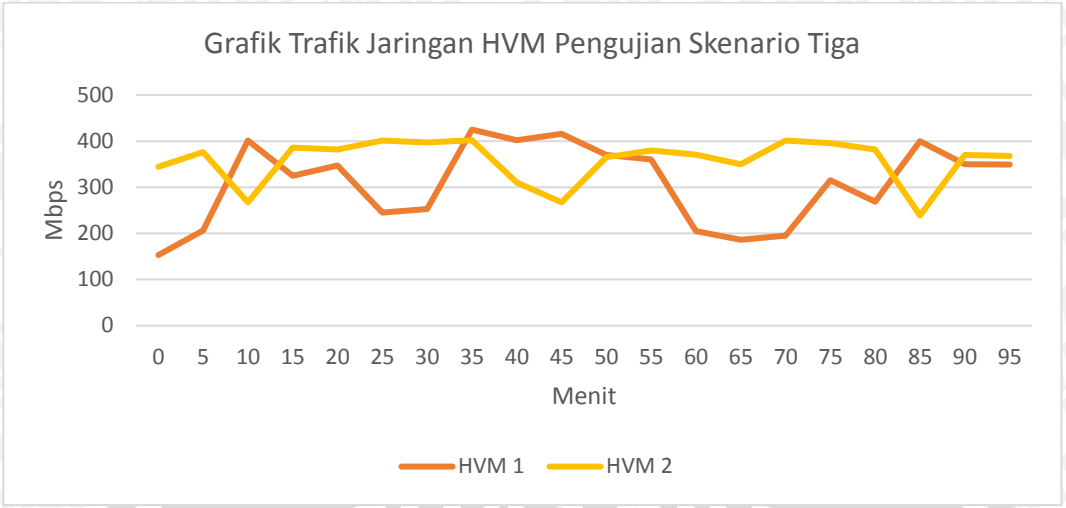
Berdasarkan hasil pengujian sistem ini juga memperlihatkan hasil bahwa beban *bandwidth* HVM A selalu dalam keadaan stabil yaitu beban *bandwidth* HVM A tidak melebihi angka 400 Mbits/sec. Hal ini membuktikan bahwa sistem mampu menyeimbangkan beban *bandwidth* serta penerapan *live migration* mesin virtual berjalan secara optimal. Perubahan-perubahan beban pada HVM A dan HVM B saat pengujian skenario satu, skenario dua, dan skenario tiga dapat dilihat dalam bentuk grafik pada gambar 5.1, gambar 5.2 dan gambar 5.3.



Gambar 5.1 Trafik Jaringan HVM Pengujian Skenario 1



Gambar 5.2 Trafik Jaringan HVM Pengujian Skenario Dua



Gambar 5.3 Trafik Jaringan HVM Pengujian Skenario Tiga

BAB 6 PENUTUP

6.1 Kesimpulan

Berdasarkan hasil penelitan yang telah dilakukan dan telah dibahas pada bab sebelumnya, maka dapat ditarik kesimpulan sebagai berikut:

1. Pada sistem yang dibangun, dari hasil pengujian proses pengimplementasian *distributed bartering algorithm* dapat berjalan dengan baik. *Distributed bartering algorithm* dapat menjadi salah satu solusi untuk mengurangi kemacetan dan *overhead* jaringan. Melihat hasil pengujian, bahwa rata-rata dari tiap skenario tidak melebihi ambang batas yaitu 400 Mbits/sec sehingga hal ini menunjukan bahwa metode yang diusulkan mampu menyeimbangkan beban trafik data antar HVM dan meminimalisir terjadi kemacetan dan *overhead* jaringan.
2. Metode negosiasi dan *swapping* antar HVM juga dapat diterapkan dengan baik. Proses *swapping* mampu menyeimbangkan kelebihan beban yang terjadi antar HVM. Dengan adanya proses *swapping* ini, trafik jaringan yang ada antar HVM dapat lebih stabil sehingga proses *swapping* dapat digunakan sebagai salah satu metode selain *live migration.*

6.2 Saran

Saran yang dapat diberikan untuk pengembangan dari sistem ini adalah diperlukannya penelitian yang lebih lanjut terkait *distributed bartering algorithm* berdasarkan kemacetan jaringan, CPU Load maupun Memory Load sehingga hasilnya nanti dapat dijadikan perbandingan dari penelitian ini.

DAFTAR PUSTAKA

Agung A. Aditya. 2012. "*Optimasi Penggunaan Sumber Daya Komputasi Di Lingkungan Cloud Computing*". Malang: PTIIK Universitas Brawijaya

Al-Kiswany, Samer, dkk, 2011."*VMFlock: Mesin virtual Co-Migration for the Cloud".* Proceedings of the 20th international symposium on High performance distributed computing, New York.

Budiyanto, Alex 2012. *E-Book Pengantar Cloud Computing*, *Cloud* Indonesia, hal 3. Furth, Borko, &Escalante, A.,2010. "*Handbook of Cloud Computing*", Springer, hal v.

Huynh Khoa, Ph.D, A. Theurer & S. Hajnoczi, 2013."*KVM Virtualized I/O Performance*".

Hwang Kai, Geoffrey C.Fox, Jack J. Dongarra, 2012. "*Distributed and Cloud Computing"*, Morgan Kaufmann, USA.

Lublin, Uri, & Ligouri, Antony, 2007. "*KVM Live Migration*" Qumranet & IBM. Microsoft-TechNet, 2012. *Hypervisor* [image online] Tersedia di:

https://blogs.technet.microsoft.com/chenley/2011/02/09/*hypervisors*/ [Diakses 20 Desember 2015]

M. R. Garey and D. S Johnson, 1979, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman.

Myint, May T. H & Thandar Thein, 2010. "*Availability Improvement in Virtualized Multiple Servers with Software Rejuvenation and Virtualization", University of Computer Studies, Yangon, Myanmar.*

Pachorkar Nilesh, R. Ingle, 2013. "*Multi-dimensional Affinity Aware VM Placement Algorithm in Cloud Computing*", International Journal of Advanced Computer Research.

Sanjay P. Ahuja, Suganya Sridharan, 2012. "*Performance Evaluation of Hypervisors for Cloud Computing",* International Journal of *Cloud* Applications and Computing, USA.

Sarna, David E.Y., 2011, Implementing and Developing *Cloud* Computing Applications. Taylor and Francis Group, LLC. Boca Raton.

Scarfone Karen, M. Suoppaya, P. Hoffman, 2011. "*Guide to Security for Full Virtualization Technologies*.

Shirley Radack, 2012. "*Cloud Computing: A Review of Features, Benefits, and Risk, and Recommendations for Secure, Efficient Implementations*", National Institute of Standarts and Technology.

Sonnek. J, Greensky. J, R. Reutiman, and A. Chandra, 2010 "Starling: Minimizing Communication *Overhead* in Virtualized Computing Platforms Using

Decentralized Affinity-Aware Migration," 39th International Conference on Parallel Processing

Stage, Alexander & Thomas Setzer, 2009. *"Network-aware migration control and scheduling of differentiated mesin virtual workloads".*Technische Universitat Munchen: Germany.

Strunk, Anja and D. Waltenegus, 2011. "*Does Live Migration of Virtual Machines Cost Energy?*". Technical University of Dresden, Dresden, Germany.

Suryatama, Indra, 2012. "*Membangun Infrastruktur Komputasi Awan Privat* Menggunakan *Ubuntu Enterprise Cloud*". Andi: Yogyakarta.

Timo Hirt, 2011. "KVM - The kernel-based virtual Machine".

LAMPIRAN

Program *Bash Shell Live Migration* (**/home/skripsi_script.sh**)

```
#!/bin/bash
#-------------------------------------------------------------------
------------
#(C) Meliza G.D Latuputty 2016
#Berkas shell ini adalah shell utama yg mengatur semua proses yg
ada di sistem
#ini.

##Input variabel (identifier)
maxhvmA=400
maxhvmB=400
tgl=`date`
migration=`cat /home/skripsi_script/migration.txt | head -n 1`

#-------------------------------------------------------------------
-------------
##traffic Bandwidth
bandwidthA=`cat /home/skripsi_script/bandwidhtA.txt | head -n 1`
bandwidthB=`cat /home/skripsi_script/bandwidhtB.txt | head -n 1`

#-------------------------------------------------------------------
-------------
##mengambil info vmload (nama vm dan traffic bandwidtht) di hvm 1
dan hvm 2
hvm2=`cat /home/skripsi_script/hvm2_vm_run.txt | head -n 1`
sleep 2

if [ $hvm2 -eq 0 ];
then
   data1=`cat /home/skripsi_script/virttophvm1.txt | head -n 1 |
tail -1`
```

```
   data2=`cat /home/skripsi_script/virttophvm1.txt | head -n 2 |
tail -1`

   data3=`cat /home/skripsi_script/virttophvm1.txt | head -n 3 |
tail -1`

   data4=`cat /home/skripsi_script/virttophvm1.txt | head -n 4 |
tail -1`

   data5=`cat /home/skripsi_script/virttophvm1.txt | head -n 5 |
tail -1`

   data6=`cat /home/skripsi_script/virttophvm1.txt | head -n 6 |
tail -1`

   data7=`cat /home/skripsi_script/virttophvm1.txt | head -n 7 |
tail -1`

   data8=`cat /home/skripsi_script/virttophvm1.txt | head -n 8 |
tail -1`

elif [$hvm2 -eq 1 ];

then

    data1=`cat /home/skripsi_script/virttophvm1.txt | head -n 1 |
tail -1`

    data2=`cat /home/skripsi_script/virttophvm1.txt | head -n 2 |
tail -1`

      data3=`cat /home/skripsi_script/virttophvm1.txt | head -n 3
| tail -1`

      data4=`cat /home/skripsi_script/virttophvm1.txt | head -n 4
| tail -1`

      data5=`cat /home/skripsi_script/virttophvm1.txt | head -n 5
| tail -1`

      data6=`cat /home/skripsi_script/virttophvm1.txt | head -n 6
| tail -1`

    data7=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 1 | tail -1`

    data8=`cat /var/lib/lib virt/images/kvmshared/virttophvm2.txt
| head -n 2 | tail -1`

elif [$hvm2 -eq 2 ];

then

    data1=`cat /home/skripsi_script/virttophvm1.txt | head -n 1 |
tail -1`

    data2=`cat /home/skripsi_script/virttophvm1.txt | head -n 2 |
tail -1`
```

```
      data3=`cat /home/skripsi_script/virttophvm1.txt | head -n 3
| tail -1`

      data4=`cat /home/skripsi_script/virttophvm1.txt | head -n 4
| tail -1`

      data5=`cat
/var/lib/libvirt/images/kvmshared/virttophvm2.txt | head -n 1 |
tail -1`

    data6=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 2 | tail -1`

    data7=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 3 | tail -1`

    data8=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 4 | tail -1`

elif [$hvm2 -eq 3 ];

then

    data1=`cat /home/skripsi_script/virttophvm1.txt | head -n 1 |
tail -1`

    data2=`cat /home/skripsi_script/virttophvm1.txt | head -n 2 |
tail -1`

      data3=`cat
/var/lib/libvirt/images/kvmshared/virttophvm2.txt | head -n 1 |
tail -1`

    data4=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 2 | tail -1`

      data5=`cat
/var/lib/libvirt/images/kvmshared/virttophvm2.txt | head -n 3 |
tail -1`

    data6=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 4 | tail -1`

    data7=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 1 | tail -1`

    data8=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 1 | tail -1`

elif [$hvm2 -eq 4 ];

then

    data1=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 1 | tail -1`

    data2=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 2 | tail -1`
```

```
      data3=`cat
/var/lib/libvirt/images/kvmshared/virttophvm2.txt | head -n 3 |
tail -1`
    data4=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 4 | tail -1`
      data5=`cat
/var/lib/libvirt/images/kvmshared/virttophvm2.txt | head -n 5 |
tail -1`
    data6=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 6 | tail -1`
    data7=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 7 | tail -1`
    data8=`cat /var/lib/libvirt/images/kvmshared/virttophvm2.txt
| head -n 8 | tail -1`
fi


####################
if [ $data1 == "vm_1" ];
then
      bandwidth-vm_1=$data2;
elif [ $data3 == "vm_2" ];
then
      bandwidth-vm_2=$data4;
elif [ $data5 == "vm_3" ];
then
      bandwidth-vm_3=$data6;
elif [ $data7 == "vm_4" ];
then
      bandwidth-vm_4=$data8;
fi


##############
bandwidth-vm_1=${bandwidth-vm_1/.*}
bandwidth-vm_2=${bandwidth-vm_2/.*}
bandwidth-vm_3=${bandwidth-vm_3/.*}
```

```
bandwidth-vm_4=${bandwidth-vm_4/.*}


#main
# INI PROSES MIGRASI
if [ $bandwidth-vm_1+$bandwidth-vm_2 -gt $maxhvmA ]
then
      totalvm=`virsh list | awk 'END {print NR-2}'`
     if [ $totalvm -eq 2 ];then
           if [ $bandwidth-vm_1 -lt $maxhvmB - ($bandwidth-
vm_3+$bandwidth-vm_3)] then;
                 elif [ $bandwidth-vm_2 -lt $maxhvmB -
($bandwidth-vm_3+$bandwidth-vm_3)] then;
                 vmtarget=1
                 else
                 vmtarget=2
           fi
           #ether-wake 08:00:27:7C:0F:71
       #sleep 10
       #echo "$vmtarget" > '/home/skripsi_script/hvm2_vmname.txt'
       echo "1" > '/home/skripsi_script/hvmA_vmrun.txt'
       echo "$vmtarget" > '/home/skripsi_script/vmmigrate.txt'
       echo "1" > '/home/skripsi_script/migration.txt'
       echo "3" > '/home/skripsi_script/hvm2_vm_run.txt'
       echo "1" > '/home/skripsi_script/totalvm_hvmA_run.txt'
       echo -en $tgl "\tMIGRATION (VM_1 to HVMB) \t\tHVMA
TRAFFIC(%) ="$bandwidthA"\tHVMB TRAFFIC(%)
="$bandwidthB"\t\tTRAFFIC_VM1(Mbps) = "$bandwidth-
vm_1"\tTRAFFIC_VM2(Mbps) = "$bandwidth-vm_2"\tTRAFFIC_VM3(Mbps) =
"$bandwidth-vm_3"\tTRAFFIC_VM4(Mbps) = "$bandwidth-vm_4"\t\n" >>
'/home/skripsi_script/log.txt'
           if [ $vmtarget -eq 1 ];then
        virsh migrate --live vm_1
qemu+ssh://kvmmode2.localdomain/system
        else
```

```
        virsh migrate --live vm_2
qemu+ssh://kvmmode2.localdomain/system

      fi

      exit

# INI PROSES SWAP

#//jika B tidak tersedia untuk migrasi, cek B yang cocok untuk
SWAP


     elif

          if ($bandwidth-vm_1 + $bandwidth-vm_4 -lt $maxhvmB &&
$bandwidth-vm_2 + $bandwidth-vm_3 -lt $maxhvmBA) { //cek SWAP VM1
dg VM3

                then ;

                virsh migrate --live vm_3
qemu+ssh://kvmmode2.localdomain/system

                virsh --connect
qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \ migrate --live
vm_$vmmigrate qemu+ssh:///system



           elif ($bandwidth-vm_1 + $bandwidth-vm_3 -lt $maxhvmB
&& $bandwidth-vm_2 + $bandwidth-vm_4 < $maxhvmA) { //cek SWAP VM1
dg VM4

                then;

                virsh migrate --live vm_1
qemu+ssh://kvmmode2.localdomain/system

                virsh --connect
qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \ migrate --live
vm_$vmmigrate qemu+ssh:///system



           elif ($bandwidth-vm_2 + $bandwidth-vm_4 -lt $maxhvmB
&& $bandwidth-vm_1 + $bandwidth-vm_3 -lt $maxhvmA) { //cek SWAP
VM2 dg VM3

                then;

                #$vm2 = "(swap dg VM3 ke B)";

                #$vm3 = "(swap dg VM2 ke A)";

                virsh migrate --live vm_2
qemu+ssh://kvmmode2.localdomain/system
```

```
                virsh --connect
qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \ migrate --live
vm_$vmmigrate qemu+ssh:///system

          elif ($bandwidth-vm_2 + $bandwidth-vm_3 -lt $maxhvmB
&& $bandwidth-vm_1 + $bandwidth-vm_4 -lt $maxhvmA) { //cek SWAP
VM2 dg VM4

                #$vm2 = "(swap dg VM4 ke B)";

                #$vm4 = "(swap dg VM2 ke A)";

                virsh migrate --live vm_4
qemu+ssh://kvmmode2.localdomain/system

                virsh --connect
qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \ migrate --live
vm_2 qemu+ssh:///system

          fi

     fi


#//jika B high traffic-----------------------------------------
-

         if ($bandwidth-vm_3 + $bandwidth-vm_4 -gt $maxB)

         then;

#//cek ketersediaan A untuk B

     if ($bandwidth-vm_3 -lt $maxhvmA - ($bandwidth-vm_1 +
$bandwidth-vm_2)) { //cek VM3 untuk migrasi

          #$vm3 = "migrasi ke A";

          then;

          virsh --connect
qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \ migrate --live
vm_3 qemu+ssh:///system

     } elif ($bandwidth-vm_4 -lt $maxhvmA - ($bandwidth-vm_1 +
$bandwidth-vm_2)) { //cek VM4 untuk migrasi

          #$vm4 = "migrasi ke A";

          then;

          virsh --connect
qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \ migrate --live
vm_4 qemu+ssh:///system

        fi

      fi
```

```
        //jika A tidak tersedia untuk migrasi, cek A yang cocok
untuk SWAP
      elif {
           if ($bandwidth-vm_3 + $bandwidth-vm_2 -lt $maxhvmA &&
$bandwidth-vm_4 + $bandwidth-vm_1 -lt $maxhvmB) { //cek SWAP VM3
dg VM1
                  then;
                  #$swap1 = $vm1; $swap2 = $vm3;
                  #$vm3 = "(swap dg VM1 ke A)";
                  #$vm1 = "(swap dg VM3 ke B)";
                  virsh --connect
qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \ migrate --live
vm_1 qemu+ssh:///system
                  virsh migrate --live vm_3
qemu+ssh://kvmmode2.localdomain/system
           elif ($bandwidth-vm_3 + $bandwidth-vm_1 -lt $maxhvmA
&& $bandwidth-vm_4 + $bandwidth-vm_2 -lt $maxhvmB) { //cek SWAP
VM3 dg VM2
                  then;
                  #$vm3 = "(swap dg VM2 ke A)";
                  #$vm2 = "(swap dg VM1 ke B)";
                  virsh --connect
qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \ migrate --live
vm_1 qemu+ssh:///system
                  virsh migrate --live vm_3
qemu+ssh://kvmmode2.localdomain/system
           elif ($bandwidth-vm_4 + $bandwidth-vm_2 -lt $maxhvmA
&& $bandwidth-vm_3 + $bandwidth-vm_1 -lt $maxhvmB)  //cek SWAP
VM4 dg VM1
                  then;
                  #$vm4 = "(swap dg VM1 ke A)";
                  #$vm1 = "(swap dg VM4 ke B)";
                  virsh --connect
qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \ migrate --live
vm_1 qemu+ssh:///system
                  virsh migrate --live vm_4
qemu+ssh://kvmmode2.localdomain/system
```

```
            elif ($bandwidth-vm_4 + $bandwidth-vm_1 -lt $maxhvmA
&& $bandwidth-vm_3 + $bandwidth-vm_2 -lt $maxhvmB) { //cek SWAP
VM4 dg VM2

                then;

                #$vm4 = "(swap dg VM2 ke A)";

                #$vm2 = "(swap dg VM4 ke B)";

                virsh --connect
qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \ migrate --live
vm_2 qemu+ssh:///system

                virsh migrate --live vm_4
qemu+ssh://kvmmode2.localdomain/system

            fi

    fi

fi


# INI PROSES REMIGRASI



             elif [ $totalvm -eq 1 ];then

       echo -en $tgl "\tTIDAK ADA MIGRATION (1VM)\t\tHVMA
TRAFFIC(%) ="$bandwidthA"\tHVMB TRAFFIC(%)
="$bandwidthB"\t\tTRAFFIC_VM1(Mbps) = "$bandwidth-
vm_1"\tTRAFFIC_VM2(Mbps) = "$bandwidth-vm_2"\tTRAFFIC_VM3(Mbps) =
"$bandwidth-vm_3"\tTRAFFIC_VM4(Mbps) = "$bandwidth-vm_4"\t\n" >>
'/home/skripsi_script/log.txt'

       exit

  else

    totalvm=`virsh list | awk 'END{print NR-2}'`

    echo -en "Total VM = "$totalvm"\n\n"

    if [ $totalvm -eq 2 ];then

       echo -en $tgl "\tTIDAK ADA MIGRATION (2VM)\t\tHVMA
TRAFFIC(%) ="$bandwidthA"\tHVMB TRAFFIC(%)
="$bandwidthB"\t\tTRAFFIC_VM1(Mbps) = "$bandwidth-
vm_1"\tTRAFFIC_VM2(Mbps) = "$bandwidth-vm_2"\tTRAFFIC_VM3(Mbps) =
"$bandwidth-vm_3"\tTRAFFIC_VM4(Mbps) = "$bandwidth-vm_4"\t\n" >>
'/home/skripsi_script/log.txt'

       exit

    elif [ $totalvm -eq 1 ];then
```

```
      hvm_vmrun=`cat /home/skripsi_script/hvm_vmrun.txt | head -n
1`

      vmmigrate=`cat /home/skripsi_script/vmmigrate.txt | head -n
1`

      let bandwidthtotal=$bandwidth-vm_1+$bandwidth-vm_2

      if [ $bandwidthtotal -gt $maxhvmA ];then

          echo -en $tgl "\tTIDAK ADA MIGRATION (1VM)\t\tHVMA
TRAFFIC(%) ="$bandwidthA"\tHVMB TRAFFIC(%)
="$bandwidthB"\t\tTRAFFIC_VM1(Mbps) = "$bandwidth-
vm_1"\tTRAFFIC_VM2(Mbps) = "$bandwidth-vm_2"\tTRAFFIC_VM3(Mbps) =
"$bandwidth-vm_3"\tTRAFFIC_VM4(Mbps) = "$bandwidth-vm_4"\t\n" >>
'/home/skripsi_script/log.txt'

          exit

        else

          echo -en $tgl "\tREMIGRATION (VM"$vmmigrate" to
HVMA)\t\tHVMA TRAFFIC(%) ="$bandwidthA"\tHVMB TRAFFIC(%)
="$bandwidthB"\t\tTRAFFIC_VM1(Mbps) = "$bandwidth-
vm_1"\tTRAFFIC_VM2(Mbps) = "$bandwidth-vm_2"\tTRAFFIC_VM3(Mbps) =
"$bandwidth-vm_3"\tTRAFFIC_VM4(Mbps) = "$bandwidth-vm_4"\t\n" >>
'/home/skripsi_script/log.txt'

          echo "2" > '/home/skripsi_script/totalvm_run.txt'

          echo "1" > '/home/skripsi_script/migration.txt'

          echo "0" >
'/home/skripsi_script/hvm'$hvm_vmrun'_vm_run.txt'

          echo "2" > '/home/skripsi_script/hvm_vmrun.txt'

        virsh --connect
qemu+ssh://kvmmode$hvm_vmrun.localdomain/system \ migrate --live
vm_$vmmigrate qemu+ssh:///system

        #sleep 10

        #ssh root@kvmmode$hvm_vmrun.localdomain pm-suspend

          exit

        fi

    fi

  fi

 fi



fi
```