BAB II
DASAR TEORI

2.1 KAJIAN PUSTAKA

Pada kajian pustaka ini membahas tentang penelitian yang telah dilakukan beserta penelitian yang akan dilakukan. Terdapat beberapa jurnal yang digunakan sebagai acuan. Penelitian Fernando, dkk. [4] tahun 2019 dengan judul *"Live Migration Ate My VM: Recovering a Virtual Machine after Failure of Post-Copy Live Migration"*. Penelitian ini bertujuan untuk mengukur hasil pemulihan pada VM saat terjadi kegagalan pada proses *live migration* menggunakan metode *postcopyFT.* Parameter yang diukur dalam penelitian ini adalah total *migration time, downtime, replication time, application performance degradation, network bandwidth degradation,* dan *filover time*. Hasil dari penelitian ini adalah nilai *downtime* untuk memigrasikan VM yang menganggur menggunakan metode *postcopyFT* berkisar antara 1.1-1.9 detik, sedangkan untuk *downtime* dengan menggunakan metode *post-copy* didapatkan nilai 9-11.6 detik. Ini terjadi karena VM berhenti terus menerus pada setiap interval *checkpoint* (100µ) ke status memori *checkpoint*. Implementasi *postcopyFT* menggunakan platform KVM/QEMU.

Penelitian Gilang, dkk. [7] tahun 2020 pada penelitian ini menerapkan konsep *live migration* pada *cloud computing* dilakukan menggunakan *virt-manager*. Hasil penelitian ini menggunakan skenario bermain *game* dengan ukuran memori 4GB, menghasilkan *migration time* yang paling rendah yaitu 10 detik dengan rata-rata *downtime* 89 mili detik. Menggunakan metode *post-copy* menghasilkan *migration time* yang rendah karena penggunaan CPU sebesar 84% dengan penggunaan memori 2279MB.

Penelitian Altahat, dkk. [8] tahun 2018 pada penelitian ini membandingkan tiga metode dalam *live migration* pada VM antara *pre-copy, post-copy,* dan *hybrid-copy* untuk *downtime,* total waktu migrasi, dan total data transfer pada saat proses migrasi berlangsung. Pada penelitian ini mengimplementasikan tiga metode menggunakan MATLAB dan mengevaluasi kinerja metode migrasi ini untuk parameter eksperimental yang berbeda. Penelitian ini bertujuan untuk mengusulkan metode mana yang cocok untuk *live migration* dari VM berdasarkan kinerjanya.

Penelitian Chou, dkk. [9] tahun 2019 pada penelitian ini menerapkan konsep *live migration* menggunakan metode *post-copy* berbasis *Fabric-Attached Memory (FAM)*. FAM adalah bagian dari arsitektur sistem yang menggunakan *fabric* jaringan berkinerja tinggi untuk menghubungkan *Node Version Manager* (NVM) di antara berbagai node dalam rak dan menghasilkan kumpulan NVM tunggal yang digunakan bersama. Sistem yang digunakan untuk melakukan *live migration* dengan menggunakan sistem *Checkpoint/Restore in Userspace (CRIU)*, CRIU merupakan alat pos pemeriksaan sumber terbuka *Linux* yang menyimpan status saat ini dari aplikasi yang sedang berjalan ke perangkat penyimpanan lokal dan memulai ulang aplikasi kapan pun diperlukan. Parameter yang dibahas dalam penelitian ini adalah waktu sibuk dan total waktu migrasi. Hasil pengujian *post-copy* berbasis FAM mendapatkan peningkatan 15% total waktu migrasi. Sedangkan untuk waktu sibuk didapatkan peningkatan 33% dan dapat membuat aplikasi yang dimigrasikan 12% lebih baik salama proses migrasi berlangsung.

Penelitian Muthiah, dkk. [6]tahun 2019 kajian ini mengimplementasikan *live migration* berbasis *proxmox* menggunakan metode *pre-copy*. Konsep *live migration* pada penelitian ini menggunakan *cloud computing* dan menggunakan *proxmox* sebagai media migrasinya. Parameter yang diukur dalam kajian ini adalah data transfer, *downtime*, dan *migration time*. Skenario pengujian dalam kajian ini menggunakan *youtube*, dimana penulis memutar video di *youtube* saat migrasi berlangsung. Pada saat pengujian migrasi *youtube* tersebut, pengujian ini sama seperti pengujian migrasi VM. Ukuran data *file* yang di migrasikan langsung dari *youtube* yang sedang di akses, sehingga mengakibatkan pengukuran yang jauh lebih besar signifikan dalam *bytes*. Pada pengujian migrasi ini, dilakukan sebanyak tiga puluh kali migrasi pada setiap VM yang mengakibatkan butuh waktu kurang lebih 2 jam dalam setiap migrasi VM. Pada VM 100 mendapatkan rata-rata *downtime* 342 ms dengan waktu migrasi 53 menit, pada VM 101 mendapatkan rata-rata *downtime* 382 ms dengan waktu migrasi 30 menit dan pada VM 103 mendapatkan rata-rata *downtime* 337 ms dengan waktu migrasi 31 menit. Hasil pengukuran bisa berbeda di ketiga VM, sehingga menjadikan *downtime* besar dengan ukuran data transfer menjadi besar dengan waktu migrasi menjadi lama. Ukuran data transfer saat di migrasikan akan semakin besar dengan pengaruh *migration speed* atau

bandwidth. Pada Tabel 2.1 merupakan perbandingan antara penelitian sebelumnya yang relevan pada penelitian saat ini. Perbandingan ini meliputi metode, mesin, dan hasil.

Tabel 2.1 Rangkuman keterkaitan penelitian dengan kajian Pustaka.

No	Peneliti	Tahun	Metode	Mesin	Hasil
1	Fernando, dkk.	2019	*Post-copy*	VM	Hasil rata-rata *migration time* 543 s. Hasil rata-rata *downtime* sangat bagus sebesar 2,2 ms.
2	Gilang, dkk.	2020	*Post-copy*	VM	Hasil rata-rata *migration time* yang paling bagus pada skenario *streaming youtube* sebesar 12 s. hasil rata-rata *downtime* yang paling bagus pada skenario menonton video *offline* didapatkan sebesar 53 ms. Sedangkan hasil rata-rata data transfer 64 GB.
3	Altahat, dkk.	2018	*Pre-copy*, *post-copy*, *hybrid-copy*	VM	Metode migrasi *Hybrid-copy* menggabungkan yang terbaik dari *Pre-copy* dan *Post-copy*. Metode ini menghasilkan migration time yang lebih rendah daripada metode *Post-copy*, dan *downtime* serta total data yang ditransfer lebih rendah daripada metode *Pre-copy*.
4	Chou, dkk	2019	*Post-copy*	*Docker container*	Memodifikasi metode *post-copy* menggunakan FAM

No	Peneliti	Tahun	Metode	Mesin	Hasil
4					(*Fabric-Attached Memory*) mendapatkan peningkatan pada *migration time* 15%, dan *busy time* 33% dari pada hanya menggunakan *post-copy* saja.
5	Muthiah, dkk.	2019	*Pre-copy*	VM	VM 100 hasil rata-rata *downtime* 342 ms dan *migration time* 53 menit, VM 101 hasil rata-rata *downtime* 382 ms dan waktu migrasi 30 menit dan pada VM 103 hasil rata-rata *downtime* 337 ms dan waktu migrasi 31 menit.

2.2 *LIVE MIGRATION*

Live migration adalah teknologi dan fitur penting dalam *virtualisasi* pusat data. Dengan migrasi langsung, VM dapat dipindahkan dari satu *host* fisik ke *host* lain dengan sedikit atau tanpa dampak pada ketersediaan aplikasi yang sedang berjalan. Ini mencegah aplikasi yang sedang berjalan agar tidak terpengaruh oleh masalah *server* fisik secara keseluruhan, sehingga meningkatkan ketersediaan layanan secara signifikan. Dalam jaringan komputer, salah satu jenis sistem komputer yang menawarkan layanan disebut *server*. *Server* memiliki prosesor yang kuat dan RAM yang besar, serta menjalankan sistem operasi khusus yang disebut sistem operasi jaringan [10]. Protokol TCP/IP digunakan untuk mengirim lalu lintas migrasi langsung melalui jaringan *Ethernet* yang menghubungkan *server cluster*. Konten yang harus dimigrasikan terutama adalah *cache CPU*, memori, dan *buffer*; namun, konten memori harus dimigrasikan secara massal. Sebagian besar menganggap bahwa *cache CPU* dan konten *buffer* hampir dapat diabaikan dibandingkan dengan konten memori [11]. Dalam migrasi langsung, klien tidak terlibat dan terus menggunakan layanan seperti biasa, tanpa terpengaruh oleh pemindahan *server* [6].

Live migration mengharuskan layanan yang masih berjalan di VM yang dipindahkan tidak terganggu, tetapi migrasi tidak-langsung tidak dibatasi oleh kebutuhan ini. Tiga kendala berikut harus diatasi untuk memigrasikan VM:

1) Migrasi data memori

   Semua status berjalan harus ditransfer ke tujuan agar VM yang dimigrasikan dapat terus beroperasi dari titik penangguhan setelah migrasi. Data status mencakup status perangkat, status memori, status CPU, dan lainnya. Migrasi data memori adalah bagaimana status operasi sering ditransfer.

2) Migrasi data penyimpanan

   Disk virtual dari VM yang dimigrasikan juga akan ditransfer ke situs tujuan jika sistem penyimpanan pusat data sumber tidak dapat diakses oleh *server* target dari migrasi VM. Hal ini karena mengakses data *disk* dari jarak jauh tidak hanya menyebabkan latensi I/O *disk* yang signifikan tetapi juga dapat melanggar *Services Level Agreement* (SLA).

3) Kontinuitas koneksi jaringan

   Beberapa taktik diperlukan untuk membuat VM dapat diakses oleh penggunanya setelah dipindahkan. Koneksi terbuka juga harus aktif selama migrasi agar dapat aktif [12].

Beberapa kegunaan *live migration* untuk *Virtualized clusters load balance, power saving,* dan *fault tolerance depend on live migration feature*, yaitu:

1) *Virtualized clusters load balance*, Pemetaan alokasi antara VM dan mesin aktual diperbarui secara berkala menggunakan *live migration*. Pembaruan ini didasarkan pada pemanfaatan mesin fisik untuk menjaga pemanfaatan *server* fisik yang seimbang di seluruh *cluster* dan mencegah kemacetan.
2) *Power saving,* Untuk mengurangi jumlah *server* fisik yang aktif dan mengalihkan *server* lain yang menganggur ke mode tidur selama jam pemanfaatan rendah, *live migration* digunakan untuk menggabungkan VM ke mesin fisik yang lebih sedikit.
3) *Fault tolerance depend on live migration feature*, mempertahankan dua salinan VM satu di *host* sumber dan satu lagi di *host* target antara setidaknya dua *server* fisik. Oleh karena itu, VM sekunder pada *host*

sekunder akan mengambil alih dan berfungsi sebagai VM primer baru jika VM primer pada *host* utama gagal [13].

Berikut parameter yang diuji dalam *live migration*:

1) Data transfer

Data Transfer adalah ukuran total data yang ditransfer selama migrasi, meskipun dibatasi oleh ukuran VM dan bergantung pada teknik migrasi yang digunakan [14].

2) *Downtime*

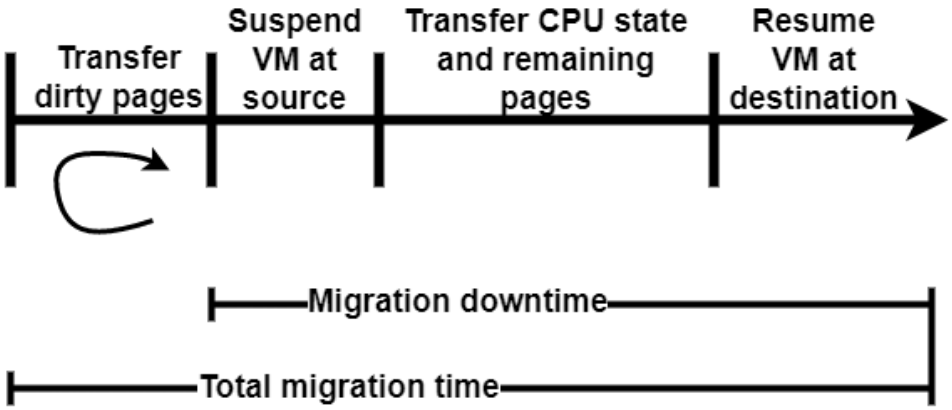
Downtime adalah waktu henti pada saat proses migrasi [6]. *Downtime* terbentuk dari waktu konfigurasi jaringan dan waktu untuk menyalin status cpu ke *host* tujuan [8].

3) *Migration time*

Migration time adalah jumlah waktu yang diperlukan untuk memindahkan mesin *virtual* dari satu *host* ke *host* lainnya [8].

2.2.1 *Pre-copy*

Pre-copy adalah teknik migrasi langsung VM yang populer, yang telah diimplementasikan pada banyak jenis *hypervisor*, seperti KVM, *VMware*, dan *Xen* [15]. *Pre-copy* dalam memetodekan *live migration* sebelum VM dimigrasikan secara iteratif pada transfer halaman memori yang diperbarui dengan memigrasikan citra VM yang konsisten, sehingga dapat lebih efisien [6].



Gambar 2.1 Mekanisme *Pre-copy* [16].

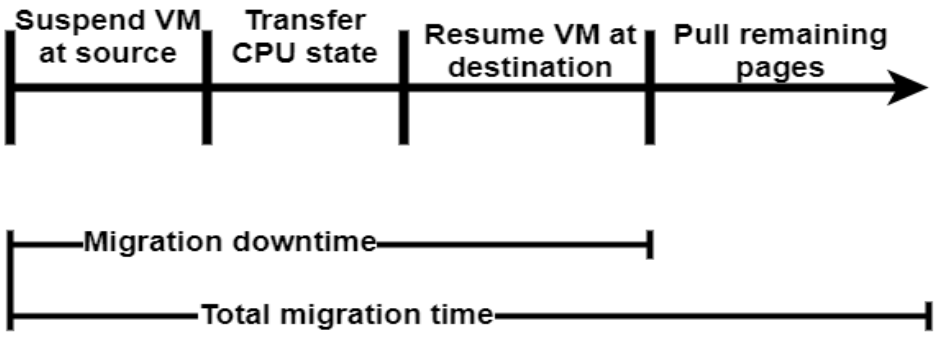
Pada Gambar 2.1 mekanisme *pre-copy* akan di jelaskan sebagai berikut:

1) *Transfer Dirty pages* merujuk pada bagian dari memori yang telah mengalami perubahan sejak awal dimuat ke dalam memori fisik. *Dirty pages* ini perlu ditransfer ulang karena tujuan akan memiliki versi *pages* yang salah. Untuk mencapai hal ini, algoritme bergerak ke fase iteratif, di mana halaman yang dikotori pada iterasi sebelumnya ditransfer ulang, hingga jumlah halaman yang tersisa untuk ditransfer berada di bawah ambang batas tertentu atau jumlah maksimum iterasi tercapai.
2) Pada awal fase kedua VM sumber dihentikan untuk memungkinkan transfer *pages* terakhir yang tersisa tanpa ada *dirty pages*. Pada fase ini *migration time* dimulai.
3) Terakhir, status CPU ditransfer dan eksekusi pada *server* asal menuju *server* tujuan.
4) Kemudian VM dilanjutkan pada host tujuan [16]. Pada fase yang terakhir total *migration time* didapatkan selama proses *live migration* berlangsung.

Pre-copy meminimalkan dua metrik yang baik yaitu *downtime* dan *application degradation* pada saat VM menjalankan beban kerja yang besar. Namun, bahkan beban kerja dengan intensitas sedang dapat mengurangi keefektifan *pre-copy* selama migrasi karena *pages* yang dikirim berulang kali [17].

2.2.2 *Post-copy*

Post-copy merupakan salah satu metode *live migration* yang popular. Saat menggunakan *post-copy*, VM dihentikan terlebih dahulu, status CPU-nya dan status eksekusi lainnya disalin, dan VM kemudian diizinkan untuk melanjutkan operasi pada *host* tujuan saat halaman memori sedang diambil [8].



Gambar 2.2 Mekanisme *Post-copy* [17].

Pada Gambar 2.2 mekanisme *post-copy* akan di jelaskan sebagai berikut:

1) *Suspend VM at source*, adalah proses penangguhan atau penghentian sementara VM pada host sumber sebelum dilakukan migrasi dan pada saat inilah terjadi *downtime.*
2) *Transfers CPU state,* merujuk pada proses transfer status CPU dari VM pada host sumber ke host tujuan. Status CPU meliputi register dan informasi lain yang diperlukan untuk melanjutkan eksekusi mesin virtual pada host tujuan setelah migrasi.
3) *Resume VM at destination*, merupakan proses melanjutkan eksekusi mesin virtual pada host tujuan setelah proses migrasi selesai. Setelah mesin virtual ditangguhkan pada host sumber dan status CPU serta informasi lainnya ditransfer ke host tujuan, mesin virtual akan dilanjutkan pada host tujuan.
4) *Pull remaining pages,* merupakan proses menyalin halaman memori yang masih "kotor" atau belum ditransfer dari *host* sumber ke *host* tujuan setelah VM ditangguhkan pada *host* sumber dan status CPU [7].

Post-copy live migration memiliki dua keunggulan utama dibandingkan dengan *Pre-copy live migration,* antara lain:

1) Fleksibilitas: Metode *post-copy* memungkinkan VM untuk mulai berjalan pada *host* tujuan segera setelah migrasi dimulai. Ini memungkinkan pengguna untuk mengakses dan menggunakan VM dengan cepat tanpa menunggu migrasi selesai sepenuhnya
2) *Total migration time* lebih rendah karena *pages* tidak diteruskan berulang pada *host* sumber selama proses migrasi berlangsung [18].

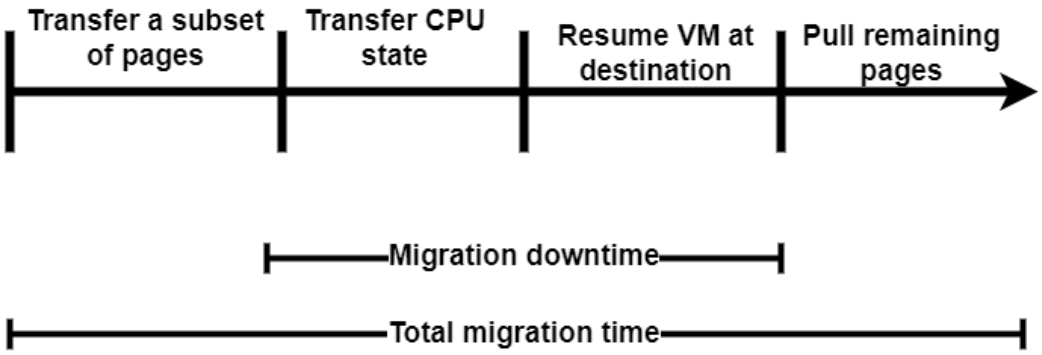
Meskipun memiliki keunggulan-keunggulan tersebut, metode post-copy juga memiliki beberapa kelemahan yang perlu diperhatikan:

1) Ketergantungan pada jaringan: Metode *post-copy* membutuhkan koneksi jaringan yang stabil dan cepat antara host sumber dan tujuan. Jika koneksi jaringan tidak memadai, proses migrasi dapat menjadi lambat atau bahkan gagal.
2) Penggunaan bandwidth yang tinggi: Metode *post-copy* melibatkan transfer sejumlah besar data dari *host* sumber ke *host* tujuan. Hal ini dapat menyebabkan penggunaan *bandwidth* yang tinggi pada jaringan, terutama jika VM memiliki memori yang besar atau beban kerja yang intensif.

3) Kemungkinan kehilangan data: Selama proses migrasi, ada kemungkinan kehilangan data jika ada perubahan yang terjadi pada VM saat migrasi sedang berlangsung. Misalnya, jika VM menulis ke halaman memori yang belum ditransfer ke *host* tujuan, data tersebut dapat hilang [7].

2.2.3 *Hybrid-copy*

Teknik *hybrid-copy* memiliki beberapa algoritma yang mencoba menggabungkan langkah-langkah *pre-copy* dan *post-copy* untuk mendapatkan migrasi yang lebih kuat dengan waktu migrasi yang lebih singkat. Salah satu algoritme ini pertama-tama melakukan migrasi [19].



Gambar 2.3 Mekanisme *Hybrid-copy* [16].

Pada Gambar 2.3 mekanisme *hybrid-copy* akan di jelaskan sebagai berikut:

1) *Transfer a subset of pages,* merupakan proses mentransfer sebagian halaman memori dari VM pada *host* sumber ke *host* tujuan selama proses migrasi.
2) *Transfer CPU state,* merupakan proses mentransfer status CPU dari VM pada *host* sumber ke *host* tujuan selama proses migrasi [7]. Status CPU meliputi *register* dan informasi lain yang diperlukan untuk melanjutkan eksekusi VM pada *host* tujuan setelah migrasi dan pada fase ini terjadi *downtime* pada VM.
3) *Resume VM at destination,* merupakan proses melanjutkan eksekusi VM pada *host* tujuan setelah proses migrasi menggunakan metode *hybrid-copy* selesai.
4) *Pull remaining pages,* merupakan proses menyalin halaman memori yang masih "kotor" atau belum ditransfer selama fase *pre-copy* dari *host* sumber ke *host* tujuan setelah VM dilanjutkan pada *host* tujuan. Halaman memori yang "kotor" adalah halaman memori yang telah diubah atau ditulis oleh VM setelah

fase *pre-copy* selesai [20]. Total *migration time* dihitung pada saat proses *live migration* berlangsung.

Metode *hybrid-copy* memiliki keunggulan dan kelemahan dalam proses *live migration,* keunggulan dan kelehaman akan dijelaskan sebagai berikut:

A. Keunggulan

1) Dapat mengurangi waktu *downtime* dibandingkan dengan metode *pre-copy*, karena hanya sejumlah kecil state eksekusi VM yang akan dikirim ke *host* tujuan.
2) Dapat mengurangi penggunaan *bandwidth* dibandingkan dengan metode *post-copy*, karena hanya sejumlah kecil state eksekusi VM yang akan dikirim ke *host* tujuan.
3) Dapat mengurangi kemungkinan kehilangan data dibandingkan dengan metode *post-copy*, karena seluruh state eksekusi VM akan dikirim ke *host* tujuan sebelum VM dijalankan pada host tujuan [20].

B. Kelemahan

1) Membutuhkan waktu yang lebih lama dibandingkan dengan metode *post-copy*, karena seluruh state eksekusi VM harus dikirim ke *host* tujuan sebelum VM dijalankan pada host tujuan.
2) Membutuhkan penggunaan *bandwidth* yang lebih tinggi dibandingkan dengan metode *pre-copy*, karena seluruh state eksekusi VM harus dikirim ke *host* tujuan sebelum VM dijalankan pada *host* tujuan [20].

2.3 *PROXMOX VIRTUAL ENVIRONMENT (VE)*

OpenVZ dan KVM termasuk dalam distribusi Linux tervirtualisasi *Proxmox VE*, yang berbasis *Debian* (64 bit). Debian adalah sebuah sistem operasi untuk komputer yang bebas (diambil dari kata *freedom*). Sistem operasi adalah kumpulan aplikasi dan alat dasar yang dibutuhkan komputer untuk berfungsi. *Debian* menawarkan lebih dari 59000 paket tambahan, berbagai perangkat lunak bawaan yang dibundel dengan rapi untuk instalasi sederhana, selain sistem operasi [5]. Banyak *server* fisik dapat dikelola secara terpusat menggunakan *Proxmox VE*. Sebuah *Proxmox VE* terdiri dari banyak *node* (setidaknya satu master dan satu *node*) dan setidaknya satu master [21]. Salah satu sistem operasi yang mampu

membuat VM dalam bentuk *Microsoft Windows* dan *Linux* adalah *Proxmox VE* [22].

Salah satu pilihan *virtualisasi open-source* adalah *Proxmox VE*. Oleh karena itu, *proxmox VE* tidak memerlukan biaya. Karena *Proxmox VE* adalah distribusi 64-bit yang berbasis *Debian* x86_64, maka tidak dapat diinstal pada komputer 32-bit yang menggunakan arsitektur i386. *Proxmox VE* bekerja dengan sejumlah lingkungan *virtualisasi* yang berbeda, termasuk KVM dan OpenVZ. Meskipun OpenVZ tidak memerlukan virtualisasi perangkat keras, menggunakan KVM pada *proxmox VE* memerlukannya jika CPU Anda tidak mendukung fitur *virtualisasi* pada perangkat keras (*virtualisasi* perangkat keras) seperti *intel VT* atau *AMD-V* [21]. Antarmuka berbasis web yang disertakan dengan *Proxmox VE* memudahkan untuk mengonfigurasi perangkat lunak dari mana saja dan pada perangkat apa saja. Selain itu, *Proxmox VE* menyertakan baris perintah dan *REST API* untuk integrasi dengan aplikasi lain. *High Availability Cluster, Live Migration, bridged networking, flexible storage, OS template building, scheduled backup*, dan *command line tools* itulah beberapa kemampuan yang disediakan oleh Proxmox *VE* [23].

Proxmox VE menggunakan *Container Virtualization* dan *Full Virtualization*:

1) *Container Virtualization (OpenVZ)* OpenVZ adalah platform yang direkomendasikan untuk operasi server Linux. OpenVZ menghasilkan beberapa kontainer terisolasi bergaya CT/VE/VPS. Setiap kontainer bertindak dan mengeksekusi seperti halnya memiliki *stand alone server*, sebuah *container* dapat di-*reboot* secara independen dan memiliki akses *super user, IP address,* memori, proses, *file,* aplikasi, *system library* dan konfigurasi sendiri, dan dapat dimulai ulang tanpa mempengaruhi yang lain.
2) *Full Virtualization* (KVM) untuk perangkat keras berbasis x86 dengan ekstensi *virtualisasi* (*Intel* VT atau AMDV CPU), mesin *virtual* berbasis kernel menawarkan alternatif yang sepenuhnya ter-*virtualisasi*. Jaringan, *card, drive,* adapter grafis, dan perangkat lain semuanya ter-*virtualisasi* dan ditugaskan ke masing-masing VM. KVM sebanding dengan XEN, tetapi KVM dibangun di dalam *Linux* dan menggunakan penjadwal standar sistem operasi dan manajemen memori [24]. KVM mempunyai kelebihan sebagai berikut:

1. Dukungan semua untuk sistem operasi
2. Kode yang efesien
3. *Open source* dan fleksibel
4. Tidak ada biaya lisensi [25].

2.3.1 *Cluster Proxmox VE*

Proxmox VE cluster manager, sering dikenal sebagai pvecm, adalah perangkat lunak yang dapat digunakan untuk menyiapkan kumpulan *server*. Kelompok seperti ini disebut sebagai *cluster*. Tidak ada aturan keras dan cepat yang menentukan jumlah maksimum node yang dapat dimuat dalam sebuah *cluster*. Pada kenyataannya, jumlah node yang layak dapat dibatasi karena kinerja *host* atau jaringan. Keuntungan menggunakan *cluster* dalam *proxmox VE* sebagai berikut:

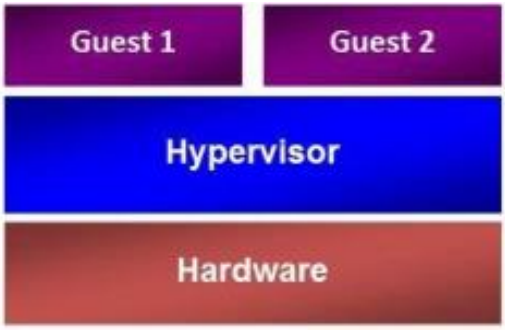
1) Manajemen web yang terpusat.
2) *Cluster multi-master*: setiap node dapat melakukan semua tugas manajemen.
3) Dapat menggunakan *live migration* vm antara node.
4) Fitur *high availability* dapat digunakan [26].

Dengan adanya sistem ini, maka perlindungan data akan meningkat dan konsistensi konfigurasi *cluster* dari waktu ke waktu akan tetap terjaga. Pengelompokan *server* juga memastikan *availability* yang lebih tinggi, *load balancing* yang tepat, dan kemampuan sistem untuk dapat menangani perubahan cepat pada beban kerja dan tuntutan pengguna [27].

2.4 *HYPERVISOR*

Hypervisor adalah bagian dari infrastruktur *virtualisasi* atau perangkat lunak khusus yang memungkinkan beberapa sistem operasi berjalan berdampingan pada mesin yang sama [1]. *Hypervisor* mengalokasikan sumber daya komputasi fisik seperti CPU dan memori ke setiap VM sesuai kebutuhan. Dengan menggunakan *hypervisor*, pengguna dapat menjalankan beberapa sistem operasi atau aplikasi secara terpisah pada satu mesin fisik, yang memungkinkan penggunaan optimal infrastruktur IT fisik [28]. Arsitektur *hypervisor* dapat dibagi menjadi 2, yaitu:

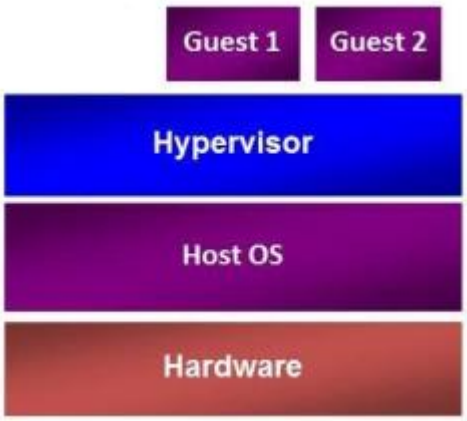
1) *Hypervisor* tipe 1 *(Bare-metal architecture)*



Gambar 2.4 Arsitektur *Hypervisor* tipe 1 [29].

Pada gamber 2.4 merupakan *hypervisor* tipe 1. *Hypervisor* jenis ini beroperasi langsung pada perangkat keras yang digunakan. Oleh karena itu, *hypervisor* dapat digunakan tanpa sistem operasi [1]. *VMware ESX/ESXi, Microsoft HyperV 2008/2012,* dan *Proxmox VE* adalah tiga contoh *hypervisor* jenis ini [23].

2) *Hypervisor* tipe 2 *(Hosted architecture)*



Gambar 2.5 Arsitektur *Hypervisor* tipe 2 [29].

Pada gambar 2.5 merupakan *hypervisor* tipe 2. *Hypervisor* tipe ni adalah program yang dapat ditambahkan ke permukaan OS standar apa pun. Sistem operasi seperti *Windows, GNU/Linux, Unix, MacOS,* dan sebagainya perlu diinstal [1]. *VMware Workstation, VMware Player, VMware Fusion, Microsoft Virtual PC*, dan *Oracle VirtualBox* adalah contoh *hypervisor* semacam ini.

Salah satu faktor yang memengaruhi kinerja mesin *virtual* adalah sistem operasi *host* atau *hypervisor*. Telah dibuktikan bahwa mesin *virtual* yang berjalan di atas *bare metal* mengungguli mesin yang berjalan di atas *hypervisor host*. Ini karena perangkat keras komputer fisik dapat diakses oleh VM. Tanpa OS yang membatasinya, VM bebas memanfaatkan sumber daya yang tersedia [23].

2.5 *VIRTUALISASI*

Virtualisasi adalah proses pembuatan salinan digital dari sumber daya fisik dalam perangkat lunak, seperti aplikasi, *server*, perangkat penyimpanan, atau koneksi jaringan [30]. Kemudian, *virtualisasi* menjadi mode terbaru dalam mengoptimalkan penggunaan *server* dan meminimalisir pemborosan. Melalui keajaiban *virtualisasi*, satu perangkat fisik dapat dibuat berfungsi seolah-olah menjadi beberapa. Hanya dua atau tiga *server* yang benar-benar dibutuhkan untuk menyelesaikan apa yang menurut banyak orang perlu dilakukan. Dengan demikian, ini sangat efektif. *Virtualisasi* dapat meningkatkan efisiensi pemanfaatan sumber daya *server* sekaligus memangkas biaya [31].

Virtualisasi melibatkan konstruksi isomorfisme yang memetakan tamu *virtual* sistem ke *host* nyata. Isomorfisme ini memetakan status tamu ke negara tuan rumah. Dari sudut sumber daya, virtualisasi membuat subset jamak secara logis dari set lengkap dari mesin. Setiap tampilan logika mungkin memiliki arsitektur yang serupa dengan tampilan fisik. Saat pengguna berin- teraksi dengan sumber daya tertentu melalui tampilan, dia tidak perlu tahu apakah tampilan itu adalah menggambar sumber daya fisik atau hanya grafik dari serangkaian sumber daya logika. Pengguna akhir tidak melihat detail sumber daya, tetapi hanya memperhatikan dan berinteraksi dengan tampilan logika disediakan oleh lapisan *virtualisasi*. Subset ini yang berperilaku sama seperti mesin nyata yang disebut VM (mesin *virtual*). Oleh karena itu, pada penyimpanan lapisan *virtualisasi* sumber daya harus diatur sebagai satukumpulan logis sumber daya yang tersedia untuk pengguna, dan pada lapisan sumber daya virtual [20]. Berikut manfaat *virtualisasi*:

1) Penggunaan sumber daya yang efisien

   Pemanfaatan perangkat keras pusat data meningkat sebagai hasil dari *virtualisasi*. Misalnya, alih-alih mempertahankan satu *server* fisik, ia dapat menyebarkan dan mengambil *server* dari kumpulan *server virtual* yang berjalan pada *server* fisik yang sama. Lebih sedikit *server* fisik yang mendasari berarti lebih sedikit permintaan untuk sumber daya pusat data seperti daya, pendinginan, dan daya cadangan.

2) Manajemen IT otomatis

Ketika komputer di-*virtualisasi*, administrasinya dapat ditangani melalui perangkat lunak. Templat mesin virtual ditentukan oleh administrator melalui program penyebaran dan konfigurasi. Dengan cara ini, kesalahan pengaturan oleh manusia dapat dihilangkan dan penyebaran infrastruktur yang berulang dapat dilakukan.

3) Pemulihan bencana yang lebih cepat

Setelah bencana alam atau serangan siber, perusahaan membutuhkan waktu beberapa hari untuk memulihkan sistem TI dan mengganti atau memperbaiki *server* fisik mereka. Namun, dalam pengaturan ter-*virtualisasi*, prosedur ini hanya membutuhkan waktu beberapa menit. Tindakan cepat ini sangat memperkuat ketangguhan dan membuka jalan bagi kelangsungan perusahaan, sehingga aktivitas rutin dapat dilanjutkan sesuai rencana.

Dengan menggunakan *virtualisasi*, seseorang dapat memperoleh manfaat dari pengaturan ter-*virtualisasi* sambil mendapatkan akses ke fitur-fitur berbagai jenis infrastruktur fisik. Ada beberapa tipe-tipe *virtualisasi* diantara lain:

1. *Virtualisasi server*

Dengan menggunakan metode yang disebut *virtualisasi server*, satu *server* fisik dapat dipecah menjadi beberapa *server virtual*. Ini adalah pendekatan yang murah dan efektif bagi bisnis untuk mengimplementasikan layanan TI mereka dan memanfaatkan sumber daya *server* mereka. Dengan tidak adanya *virtualisasi server, server* fisik akan menganggur, hanya menggunakan sebagian kecil dari total daya komputasi mereka. [32].

2. *Virtualisasi* penyimanan

Dengan *virtualisasi* penyimpanan, sistem penyimpanan fisik seperti NAS dan SAN dapat dikonsolidasikan ke dalam satu unit logis. Di pusat data, dapat menggabungkan perangkat penyimpanan dari berbagai produsen dan desain. Dengan *virtualisasi* penyimpanan, semua penyimpanan data fisik diubah menjadi kumpulan penyimpanan *virtual* yang dapat dikelola dan terperinci. Dengan menggabungkan berbagai perangkat penyimpanan jaringan ke dalam satu perangkat penyimpanan, manajer TI dapat

menyederhanakan proses penyimpanan termasuk pengarsipan, pencadangan, dan pemulihan [20].

3. *Virtualisasi* jaringan

   Semua jaringan komputer memiliki elemen perangkat keras seperti *switch*, *router*, dan *firewall*. Organisasi yang memiliki kantor di beberapa lokasi geografis bisa memiliki beberapa teknologi jaringan yang berbeda dan bekerja bersama untuk menciptakan jaringan perusahaan. *Virtualisasi* jaringan adalah proses yang menggabungkan seluruh sumber daya jaringan ini untuk memusatkan tugas administratif. Administrator dapat menyesuaikan dan mengontrol elemen ini secara *virtual* tanpa perlu menyentuh komponen fisik sehingga sangat menyederhanakan manajemen jaringan.

4. *Virtualisasi* data

   Saat ini, bisnis mendapatkan informasi dari berbagai sumber dan menyimpannya dalam berbagai format. *Cloud* dan pusat data lokal adalah dua contoh lokasi penyimpanan data yang memungkinkan. Melalui penggunaan perangkat lunak, *virtualisasi* data menyediakan lapisan perantara antara data dan program yang membutuhkannya. Perangkat lunak *virtualisasi* data menerima permintaan informasi dari aplikasi dan mengembalikan data dengan cara yang benar. Karena kebutuhan akan kemampuan beradaptasi yang lebih besar dalam integrasi data dan dorongan analisis data lintas fungsi, bisnis telah beralih ke solusi *virtualisasi* data.

5. *Virtualisasi* aplikasi

   Melalui penggunaan *virtualisasi* aplikasi, fungsionalitas inti program dapat dijalankan pada OS yang berbeda dengan OS tempat aplikasi tersebut dikembangkan. Seorang pengguna *Linux*, misalnya, tidak perlu mengubah konfigurasi mesin untuk menjalankan program *Windows*. Untuk memperoleh *virtualisasi* aplikasi, ikuti praktik berikut:

   a) *Streaming* aplikasi – Aplikasi ini di-*streaming* dari server jarak jauh dan hanya berjalan di perangkat pengguna saat diperlukan.

b) *Virtualisasi* aplikasi berbasis *server* – Tanpa menginstal apa pun, pengguna dapat mengakses aplikasi jarak jauh melalui *browser* atau antarmuka klien.

c) *Virtualisasi* aplikasi lokal – Kode sumber aplikasi disertakan dengan lingkungannya sendiri, sehingga dapat digunakan di OS apa pun tanpa modifikasi [32].

2.6 *QUALITY OF SERVICE* (QoS)

Sebuah metode yang dikenal sebagai kualitas layanan, atau QoS, digunakan untuk mengelola *throughput, delay*, *jitter*, dan *packet loss* untuk aliran dalam jaringan [33]. Qos adalah upaya untuk mendefinisikan kualitas dan sifat layanan dan teknik untuk mengukur seberapa baik kinerja jaringan. Satu set atribut kinerja yang telah ditentukan dan terhubung ke sebuah layanan diukur menggunakan QoS [34].

Dengan menjamin bahwa pengguna menerima kinerja yang konsisten dari aplikasi berbasis jaringan, QoS dimaksudkan untuk meningkatkan produktivitas bagi pengguna akhir (klien). QoS menggambarkan kapasitas jaringan untuk menggunakan berbagai teknologi untuk melayani jenis lalu lintas jaringan tertentu dengan lebih baik. Dalam jaringan berbasis IP dan *internet* secara keseluruhan, QoS merupakan kesulitan yang signifikan [33]. Parameter QoS yang digunakan sebagai berikut:

1) *Throughput*

Throughput adalah jumlah semua kedatangan paket yang berhasil diamati di tujuan selama periode waktu tertentu dibagi dengan durasi periode waktu tersebut. [34]. Dalam praktiknya, *throughput* identik dengan *bandwidth*, dan karenanya kedua istilah tersebut sering digunakan secara bergantian [33]. Menurut versi TIPHON (*Telecommunication and Internet Protocol Harmonization Over Network*) standarisasi nilai *throughput* dapat dilihat pada Tabel 2.2.

Tabel 2.2 Standarisasi *Throughput* menurut TIPHON [35].

Kategori *throughput*	*Throughput*
Buruk	0-338 kbps

Kategori *throughput*	*Throughput*
Cukup bagus	338-700 kbps
Bagus	700-1200 kbps
Lebih bagus	1200 kbps-2.1Mbps
Sangat bagus	>2.1 Mbps

Perhitungan *throughput* dapat dilihat pada persamaan (2.1).

$$Throughput = \frac{Jumlah\ data\ yang\ dikirim}{Waktu\ pengiriman\ data} \quad \textbf{(2.1)}$$

2) *Delay*

Dalam arsitektur jaringan, *delay* adalah jumlah waktu yang dibutuhkan paket untuk tiba di tempat tujuan sebagai akibat dari antrian atau memilih jalur lain untuk menghindari kemacetan [34]. *Delay* di dalam jaringan dapat digolongkan sebagai berikut:

1) *Packetization delay*

Delay yang disebabkan oleh waktu yang diperlukan untuk membuat paket IP dari data pengguna. *Delay* ini hanya terjadi pada sumber asli informasi.

2) *Queuing delay*

Waktu pemrosesan yang dibutuhkan oleh *router* untuk menangani siaran paket di jaringan adalah yang menyebabkan *delay* ini. Biasanya, *delay* ini hanya beberapa ratus mikrodetik saja.

3) *Delay* propagasi

Delay yang dikenal sebagai penundaan propagasi dihasilkan dari proses informasi yang mengalir saat melalui media transmisi, seperti kabel UTP, koaksial, atau kabel tembaga [33].

Standarisasi nilai jitter menurut TIPHON dapat dilihat pada Tabel 2.3 dan utuk melakukan perhitungan *delay* dapat menggunakan persamaan (2.2).

Tabel 2.3 Kategori *Delay* [35].

Kategori *Delay*	Besar *Delay*
Sangat bagus	<150 milidetik
Bagus	150 s/d 300 milidetik
Sedang	300 s/d 450 milidetik
Buruk	>450 milidetik

$$Rata - rata\ delay\ per\ paket = \frac{Total\ delay}{Jumlah\ paket\ yang\ diterima} \quad \textbf{(2.2)}$$

3) *Jitter*

Jitter adalah perubahan variasi *delay* selama periode waktu tertentu. *Jitter* didefinisikan sebagai variasi *delay* dari waktu ke waktu [34]. Variasi volume trafik dan jumlah tabrakan paket (*congestion*) dalam jaringan IP akan memiliki dampak yang signifikan terhadap besarnya nilai *jitter*. Kemungkinan terjadinya kongesti akan meningkat seiring dengan meningkatnya beban trafik jaringan, yang juga akan meningkatkan nilai *jitter*. Nilai QoS akan menurun ketika *jitter* meningkat. Nilai *jitter* harus dijaga seminimal mungkin untuk mendapatkan nilai QoS jaringan yang memuaskan [33]. Standarisasi nilai *jitter* menurut TIPHON dapat dilihat pada Tabel 2.4 dan utuk melakukan perhitungan *jitter* dapat menggunakan persamaan (2.3).

Tabel 2.4 Kategori *Jitter* [35].

Kategori Degradasi	***Peak Jitter***
Sangat bagus	0 milidetik
Bagus	0 s/d 75 milidetik
Sedang	76 s/d 125 milidetik
Buruk	126 > 255 milidetik

$$Rata - rata\ Jitter = \frac{Total\ variasi\ delay}{Jumlah\ paket\ yang\ diterima} \quad \textbf{(2.3)}$$

4) *Packet Loss*

Packet loss adalah sebuah indikator yang menggambarkan kondisi di mana ada paket-paket data yang tidak berhasil dikirim atau diterima selama proses transfer data. Kejadian *packet loss* ini bisa terjadi karena adanya tabrakan (*collision*) dan kepadatan (*congestion*) pada jaringan yang digunakan. Kegagalan paket tersebut mencapai tujuan, dapat disebabkan oleh beberapa kemungkinkan, diantaranya yaitu:

1) Terjadinya *overload* trafik didalam jaringan.
2) Tabrakan *(congestion)* dalam jaringan.
3) *Error* yang terjadi pada media fisik.

4) Kegagalan yang terjadi pada sisi penerima antara lain bisa disebabkan karena *overflow* yang terjadi pada *buffer* [33].

Standarisasi nilai *packet loss* menurut TIPHON dapat dilihat pada Tabel 2.5 dan rumus untuk menghitung *packet loss* dapat dilihat pada persamaan (2.4).

Tabel 2.5 Kategori *Packet Loss* [35].

Kategori Degradasi	Packet Loss
Sangat bagus	0-2%
Bagus	3-14%
Sedang	15-24%
Buruk	>25%

$$Packet\ loss = \frac{Paket\ yang\ dikirim - paket\ yang\ diterima}{Paket\ yang\ dikirim}\ x\ 100\% \quad \textbf{(2.4)}$$

Berikut beberapa model layanan QoS:

1) *Best-Effort Model*

Best-Effort Service adalah model layanan yang bertujuan untuk mengirimkan data kapan pun diperlukan, dalam jumlah berapa pun, tanpa memerlukan izin akses atau pemberitahuan sebelumnya ke jaringan. Hal ini dapat dipahami sebagai upaya jaringan untuk mengirimkan paket data sebanyak mungkin, tanpa memberikan jaminan apa pun terkait keandalan, penundaan, atau *throughput*. Segera setelah data tersedia untuk transmisi, data akan dikirim melalui media perantara kapan pun data tersebut siap dan dapat diakses. Namun, layanan ini tidak menjamin tanggung jawab atas kehilangan data atau penundaan yang signifikan di sepanjang perjalanan; jika data hilang selama transit atau mengalami penundaan yang lama, tidak ada pihak atau perangkat yang bertanggung jawab.

2) *Integrated Service Model* (IntServ)

Integrated Service Model atau IntServ adalah sebuah model layanan yang memungkinkan dukungan untuk berbagai kebutuhan *Quality of Service* (QoS). Dalam model QoS ini, aplikasi harus meminta jenis layanan tertentu dari jaringan sebelum mengirimkan data. Permintaan ini

didasarkan pada sinyal yang jelas, di mana aplikasi menginformasikan profil lalu lintas (*traffic*) jaringan dan meminta jenis layanan yang mencakup alokasi bandwidth dan pengendalian keterlambatan (*delay*). Model IntServ bertujuan untuk mengirimkan data hanya setelah mendapatkan persetujuan dari jaringan berdasarkan informasi yang diberikan, sehingga dapat mengirimkan data sesuai dengan profil lalu lintas jaringan yang telah diminta sebelumnya.

3) *Differentiated Service Model* (DiffServ)

Differentiated Service Model juga dikenal sebagai DiffServ, adalah model layanan serbaguna yang dirancang untuk mengakomodasi berbagai persyaratan *Quality of service* (QoS). Dalam model QoS ini, langkah pertama melibatkan klasifikasi semua paket yang masuk dalam jaringan. Klasifikasi ini dicapai dengan menambahkan informasi spesifik pada *header* IP setiap paket, yang didedikasikan untuk pengaturan QoS. Setelah paket diklasifikasikan pada perangkat jaringan terdekat, informasi ini digunakan oleh jaringan untuk menentukan bagaimana memperlakukan data lalu lintas. Perlakuan ini dapat melibatkan mekanisme antrian, pembentukan, dan pengaturan. Setelah proses ini, aliran data dibuat, memastikan bahwa layanan yang diberikan sesuai dengan standar QoS yang telah ditetapkan untuk para penggunanya [33].