BAB II
LANDASAN TEORI

Pada bab ini akan dibahas landasan teori dalam pembuatan *cluster* yang menggunakan openMosix. Pada bab ini juga dijelaskan tentang arsitektur *cluster* hingga keterbatasannya.

2.1 Teknologi C*luster*ing

Secara umum, saat orang membicarakan mengenai "*clustering*", mereka akan mengacu pada suatu teknologi yang memungkinkan sejumlah komputer untuk bekerja sama menyelesaikan permasalahan komputasi biasa. Permasalahan komputasi tersebut bisa berupa komputasi sains (dengan pemakaian CPU yang intensif) sampai bermacam-macam proses yang tidak memiliki kesamaan.

Teknologi *cluster* ini dibutuhkan untuk meningkatkan kinerja beberapa komputer agar menjadi suatu sistem tunggal sumber daya komputasi yang melakukan pekerjaan besar. Dari sisi pengguna, ia tak merasa bahwa pekerjaan yang dia berikan telah dibagi ke mesin fisik yang berbeda.

2.2 Arsitektur C*luster*ing

Suatu komputer yang ter-*cluster* memiliki suatu arsitektur tertentu, dimana arsitektur tersebut memungkinkan suatu komputer dapat berkomunikasi antar komputer satu dengan lainnya. Arsitektur tersebut terdiri dari dua komponen yaitu komponen *hardware* dan *software*. Sekumpulan komputer (pada suatu jaringan komputer) independen yang beroperasi dan terlihat oleh *client* jaringan seolah-olah komputer-komputer tersebut adalah satu buah unit komputer.

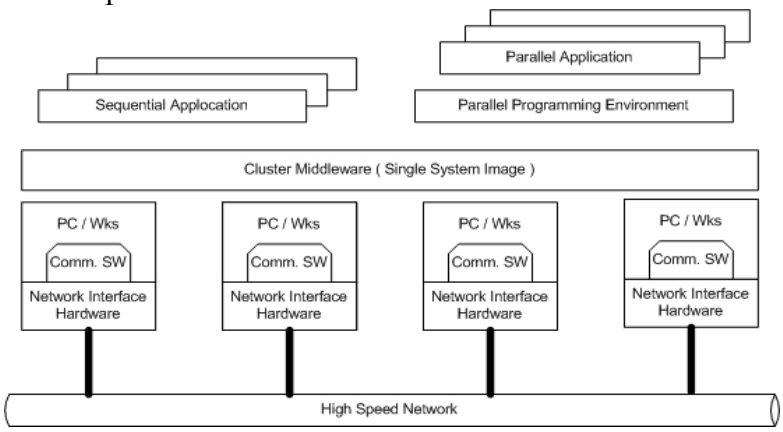
Clustering dirancang untuk meningkatkan kemampuan kinerja dari komputer-komputer yang berada pada suatu jaringan komputer untuk dapat meningkatkan hal-hal berikut:

a) Toleransi kesalahan (*fault tolerance*), yang dapat menyebabkan *node* lainnya (misal komputer B) akan mengambil alih kerja *node* utama (sebutan untuk *node* yang melakukan eksekusi program tertentu, misal komputer A) ketika *node*

tersebut mengalami kegagalan. *Client* tidak akan melihat pergantian peran ini. Dengan begitu, *downtime* pun dapat dikurangi secara drastis.

b) Penyerataan beban (*load-balancing*), yang dapat mendistribusikan beban satu *node* ke semua *node* anggota *cluster.* Dengan begitu, kinerja dan skalabilitas *node* utama pun menjadi relatif lebih baik.

Bagian terpenting dari komputer *cluster* adalah adanya sebuah aplikasi *middleware* yang dapat menggabungkan seluruh anggota dalam *cluster* sehingga dapat bekerja sama. Tugas utama dari aplikasi *middleware* ini adalah untuk komunikasi dan sinkronisasi antar komputer.



Gambar 2.1. Arsitektur Linux cluster

2.2.1. Hardware

Pada dasarnya suatu komputer memiliki komponen-komponen utama yang terdiri dari:

a) Prosessor
b) Memori
c) Motherboard
d) Hard disk
e) Network Card

Dalam membuat suatu jaringan pastilah diperlukan komponen atau peralatan tambahan, dimana komponen tersebut difungsikan untuk menghubungkan komputer satu dengan yang lainnya. Komponen tersebut adalah *switch* atau *hub,* untuk membantu peningkatan kecepatan data pada jaringan, maka lebih dipilih menggunakan switch karena switch memiliki kelebihan

dibandingkan hub, yaitu switch dapat menyimpan alamat fisik dari network card yang terpasang pada masing-masing komputer.

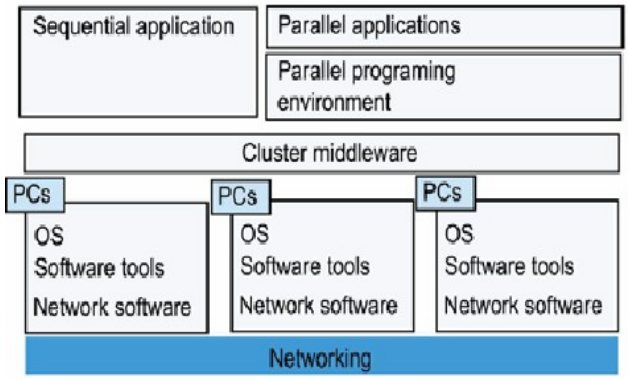


Gambar 2.2. Switch

Pada teknologi *cluster*ing, hardware yang digunakan diupayakan untuk menggunakan komputer yang memiliki spesifikasi hardware yang sama. Hal ini dimaksudkan untuk meningkatkan kinerja dari *cluster*ing tersebut. Selain itu dengan adanya hardware yang sama pada tiap-tiap komputer akan memudahkan sistem *cluster* untuk melakukan penyeimbangan beban proses (load balancing) karena tiap-tiap komputer memiliki performansi yang sama dalam mengeksekusi suatu program.

2.2.2. Software

Pada bagian software komputer pastilah harus memiliki sistem operasi. sistem operasi adalah komponen vital dalam suatu komputer. sistem operasi adalah landasan dimana suatu aplikasi akan dieksekusi. Pada *clustering* sistem operasi pada setiap komputer terkluster harus memiliki *cluster middleware*. *cluster middleware* digunakan oleh sistem operasi untuk berkomunikasi antar komputer satu dengan komputer yang lainnya seperti pada gambar dibawah :



Gambar 2.3. *Arsitektur Software pada masing-masing komputer cluster*

2.3 Jenis-jenis *cluster*

Karakteristik utama dari *cluster*ing bagaimana suatu *task* ditangani secara bersama-sama. Kata "bersama-sama" di sini yang memedakan *cluster* dengan mekanisme jaringan komputer biasa yang menjalankan program *client-server*. *clustering* difokuskan pada mekanisme sistem operasi dan hardware untuk menyatukan sumber daya setiap *node* (CPU, memori, disk, dan lain sebagainya).

Secara umum *cluster* memiliki dua jenis:

a) High Performance Computing (HPC).

Secara umum jenis *cluster* HPC ditujukan pada bagaimana suatu proses komputasi diakselerasi, dengan demikian *task* bisa diselesaikan lebih cepat.

b) High Availability (HA)

Secara umum jenis *cluster* ini ditujukan agar program yang dijalankan di atasnya bisa terus *running*, sekalipun salah satu *node hang* atau down. Contohnya adalah web server Apache yang diset dengan suatu *redirector*, sehingga jika salah satu server down, server lain bisa mengambil alih

2.4 SSI (Single System Image)

SSI adalah sistem cluster yang menggabungkan beberapa node (PC dan sebagainya) menjadi satu kesatuan sehingga dari pandangan luar (end user) terlihat sebagai satu kesatuan komputasi. "Satu kesatuan" yang dimaksud disini adalah satuan unit CPU, satuan memori, serta satuan media penyimpanan(storage).

Ada dua cara untuk membentuk satu kesatuan ini:

a) Middleware

Secara umum ini meliputi semua komponen/library yang menjadi perantara

antara program yang berjalan di *user mode* dengan kernel. Contoh: sistem *Batch Scheduler* seperti Condor atau Maui Scheduler atau Enfuzion. Library PVM dan MPI (misalnya MPICH) bisa juga dimasukan dalam kategori *middleware*, karena PVM dan MPI sebenarnya juga perantara antara program paralel dengan sistem operasi.

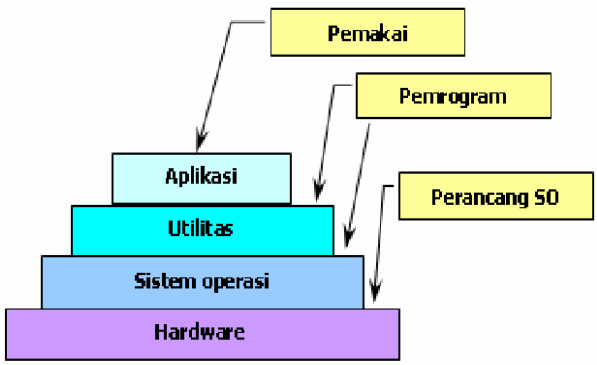
b) Modifikasi kernel (kernel extention)

Secara umum, ini meliputi semua tambahan kernel sistem operasi yang membuat sistem operasi tersebut menjadi sistem cluster, termasuk yang bertipe SSI. IBM Sysplex, Compaq Tru Cluster, VAX Cluster adalah contoh-contoh SSI yang bersifat komersial. Sedangkan openMosix adalah contoh SSI yang dilisensikan di bawah GPL (GNU General Public License).

2.5 Sistem Operasi

Saat ini banyak sekali jenis-jenis sistem operasi diantaranya Microsoft Windows, Machintos, Sun Solaris, Linux, Unix dan lain-lain. Dari kesemua sistem operasi memiliki karakteristik yang berbeda-beda.

Sistem operasi adalah suatu program yang bertindak sebagai perantara antara user dengan komputer hardware. Dimana sistem operasi dimaksudkan untuk memberikan ruang lingkup dimana user dapat mengeksekusi program. Seperti pada gambar terlihat bahwa sistem operasi menjembatani antara hardware dengan komponen aplikasi atau user.



Gambar 2.4. Sistem Operasi sebagai jembatan antara Hardware dengan Aplikasi

Sistem operasi memiliki komponen sistem yang diantaranya adalah :

1. Manajemen Proses
2. Manajemen memori utama
3. Manajemen file

4. Manajemen I/O
5. Manajemen memori sekunder

2.5.1. Sistem Operasi Linux

Linux adalah sebuah sistem operasi yang sangat mirip dengan sistem sistem UNIX, karena memang tujuan utama rancangan dari proyek Linux adalah *UNIX compatible*. Sejarah Linux dimulai pada tahun 1991, ketika mahasiswa Universitas Helsinki, Finlandia bernama Linus Benedict Torvalds menulis Linux, sebuah kernel untuk prosesor 80386, prosesor 32-bit pertama dalam kumpulan CPU Intel yang cocok untuk PC.

Pada awal perkembangannya, source code Linux disediakan secara bebas melalui Internet. Hasilnya, pengembangan Linux merupakan kolaborasi para pengguna dari seluruh dunia, semuanya dilakukan secara eksklusif melalui internet. Bermula dari kernel awal yang hanya mengimplementasikan subset kecil dari sistem UNIX, kini sistem Linux telah tumbuh sehingga memasukan banyak fungsi UNIX. Saat ini menurut LWN (Linux Weekly News – http://lwn.net/Distributions/) tercatat sekitar 420 jenis distro Linux.

Linux terdistrubusi di bawah Lisensi Publik Umum GNU (GPL), dimana peraturannya disusun oleh Free Software Foundation. Linux bukanlah perangkat lunak domain publik. *Public Domain* berarti bahwa pengarang telah memberikan *copyright* terhadap perangkat lunak mereka, tetapi *copyright* terhadap kode Linux masih dipegang oleh pengarang-pengarang kode tersebut. Linux adalah perangkat lunak bebas, namun bebas dalam arti bahwa siapa saja dapat mengkopi, memodifikasi, memakainya dengan cara apa pun, dan memberikan salinan mereka kepada siapa pun tentang larangan atau halangan.

Implikasi utama peraturan lisensi Linux adalah bahwa siapa saja yang menggunakan Linux, atau membuat modifikasi dari Linux, tidak boleh membuatnya menjadi hak milik sendiri. Jika sebuah perangkat lunak dirilis berdasarkan lisensi GPL, produk tersebut tidak boleh didistribusi hanya sebagai produk biner (*binary only*). Perangkat lunak yang dirilis atau akan dirilis tersebut harus disediakan sumber kodenya bersamaan dengan distribusi binernya.

2.5.1.1. Komponen Sistem Operasi Linux

2.5.1.1.1. Manajemen Proses

Proses adalah sebuah program yang sedang dieksekusi. Sebuah proses membutuhkan beberapa sumber daya untuk menyelesaikan tugasnya. Sumber daya tersebut dapat berupa CPU time, memori, file-file, dan perangkat- perangkat I/O. Sistem operasi mengalokasikan sumber daya-sumber daya tersebut saat proses itu diciptakan atau sedang diproses/dijalankan. Ketika proses tersebut berhenti dijalankan, sistem operasi akan mendapatkan kembali semua sumber daya yang bisa digunakan kembali. Sistem operasi bertanggung-jawab atas aktivitas-aktivitas yang berkaitan dengan managemen proses seperti:

a) Pembuatan dan penghapusan file.
b) Pembuatan dan penghapusan direktori.
c) Mendukung manipulasi file dan direktori.
d) Memetakan file ke *secondary-storage*.
e) Mem-*back-up* file ke media penyimpanan yang permanen.

2.5.1.1.2. Manajemen Memori Utama

Memori utama atau lebih dikenal sebagai memori adalah sebuah array yang besar dari word atau byte, yang ukurannya mencapai ratusan, ribuan, atau bahkan jutaan. Setiap word atau byte mempunyai alamat tersendiri. Memori utama berfungsi sebagai tempat penyimpanan instruksi/data yang akses datanya digunakan oleh CPU dan perangkat I/O. Memori utama termasuk tempat penyimpanan data yang yang bersifat volatile (tidak permanen) yaitu data akan hilang kalau komputer dimatikan. Sistem operasi bertanggung-jawab atas aktivitas-aktivitas yang berkaitan dengan managemen memori seperti:

a) Menjaga track dari memori yang sedang digunakan dan siapa yang menggunakannya.
b) Memilih program yang akan di-load ke memori.

2.5.1.1.3. Manajemen File

File adalah kumpulan informasi yang berhubungan, sesuai dengan tujuan pembuat file tersebut. Umumnya file merepresentasikan program dan data.

file dapat mempunyai struktur yang bersifat hirarkis (direktori, volume, dll.). Sistem operasi mengimplementasikan konsep abstrak dari file dengan mengatur media penyimpanan massa, misalnya tapes dan disk. Sistem operasi bertanggung-jawab dalam aktivitas yang berhubungan dengan managemen file:

a) Pembuatan dan penghapusan file.
b) Pembuatan dan penghapusan direktori.
c) Mendukung manipulasi file dan direktori.
d) Memetakan file ke *secondary-storage*.
e) Mem-*back-up* file ke media penyimpanan yang permanen.

2.5.1.1.4. Manajemen I/O

Sering disebut device manager. Menyediakan device driver yang umum sehingga operasi I/O dapat seragam (membuka, membaca, menulis, menutup). Contoh: pengguna menggunakan operasi yang sama untuk membaca berkas pada perangkat keras, CD-ROM dan floppy disk. Komponen Sistem Operasi untuk sistem I/O:

a) *Buffer*: menampung sementara data dari/ ke perangkat I/O
b) Spooling: melakukan penjadualan pemakaian I/O sistem supaya lebih efisien (antrian , dsb.)
c) Menyediakan driver: untuk dapat melakukan operasi rinci untuk perangkat keras I/O tertentu.

2.5.1.1.5. Manajemen Memori Sekunder

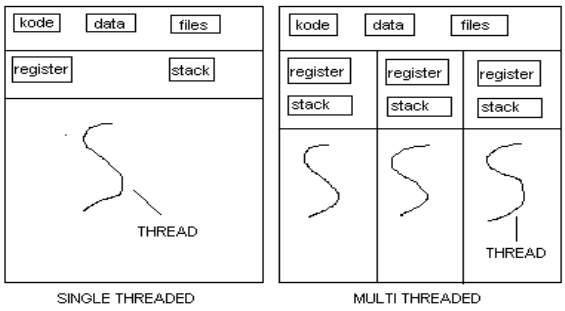
Data yang disimpan dalam memori utama bersifat sementara dan jumlahnya sangat kecil. Oleh karena itu, untuk menyimpan keseluruhan data dan program komputer dibutuhkan penyimpanan sekunder yang bersifat permanen dan mampu menampung banyak data, sebagai back-up dari memori utama. Contoh dari penyimpanan sekunder adalah hard-disk, disket, dll. Sistem operasi bertanggung-jawab atas aktivitas-aktivitas yang berkaitan dengan managemen disk seperti:

a) free-space management
b) alokasi penyimpanan
c) penjadualan disk

2.5.1.2. Thread dan Fork

1. **Thread**

Thread dalam hal ini adalah "thread of execution", didefinisikan sebagai sekumpulan instruksi yang dapat dieksekusi secara paralel dengan *thread* lainnya, dengan menggunakan metode time slice (ketika satu prosesor melakukan perpindahan antara satu thread ke thread lainnya) atau multiprocess (ketika thread-thread tersebut dieksekusi oleh prosesor yang berbeda dalam satu sistem). Thread sebenarnya mirip dengan proses, tapi cara berbagi sumber daya antara proses dengan thread adalah sangat berbeda. Multiplethread dapat dieksekusi secara paralalel pada sistem komputer. Secara umum multithreading melakukan time-slicing (sama dengan time-division multipleks), dimana sebuah prosesor bekerja pada thread yang berbeda, dimana suatu kasus ditangani tidak sepenuhnya secara simultan, untuk prosesor tunggal pada dasarnya benar-benar melakukan sebuah pekerjaan pada satu waktu. Teknik *switching* ini memungkinkan prosesor seolah-olah bekerja secara simultan.

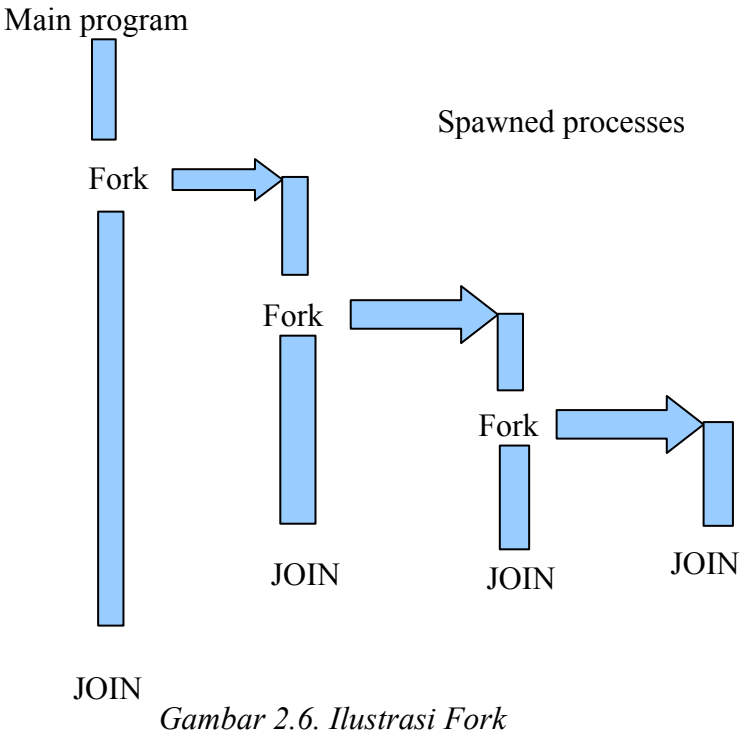


Gambar 2.5. Ilustrasi Thread

2. **Fork**

Secara teknis, seluruh pembuatan proses dibuat oleh sebuah proses (*parent*) dengan memanggil sebuah *system call* untuk pembuatan proses. System call ini memerintahkan sistem operasi untuk membuat proses baru dan mengindikasikan program apa yang akan dieksekusi. Pada sistem UNIX, system call untuk membuat proses adalah *fork.* System call ini membuat duplikat dari proses yang memanggilnya. Setelah fork, kedua proses (*parent* dan *child*) memiliki *memory image* yang sama, *environment string* yang sama dan open file yang sama, tetapi memiliki *address space* yang berbeda. Hal ini diperlukan agar kedua proses tersebut tidak saling

tergantung satu sama yang lain (independen). Setelah proses *fork* selesai, proses *child* dapat mengeksekusi *execve* untuk mengubah image memory dan menjalankan program yang baru dan tidak tergantung kepada proses *parent* .



Gambar 2.6. Ilustrasi Fork

2.6 Jaringan Komputer dan Topologi

Jaringan komputer adalah sebuah kumpulan komputer, printer dan peralatan lainnya yang terhubung. Informasi dan data bergerak melalui kabel-kabel sehingga memungkinkan pengguna jaringan komputer dapat saling bertukar dokumen dan data, mencetak pada printer yang sama dan bersama-sama menggunakan *hardware/software* yang terhubung dengan jaringan. Tiap komputer, printer atau perangkat yang terhubung dengan jaringan disebut *node*. Sebuah jaringan komputer dapat memiliki dua, puluhan, ribuan atau bahkan jutaan *node*. Sebuah jaringan biasanya terdiri dari 2 atau lebih komputer yang saling berhubungan diantara satu dengan yang lain, dan saling berbagi sumber daya misalnya *CDROM*, Printer, pertukaran data atau memungkinkan untuk saling berkomunikasi secara elektronik. Komputer yang terhubung tersebut, dimungkinkan berhubungan dengan media kabel, saluran telepon, gelombang radio, satelit, atau sinar infra merah.

2.6.1. Jenis Jaringan Komputer

Jaringan komputer dapat dibedakan berdasarkan cakupan geografisnya, ada 4 kategori utama jaringan komputer, diantaranya:

1. LAN (Local Area Network)
2. MAN (Metropolitan Area Network)
3. WAN (Wide Area Network)
4. GAN (Global Area Network)

2.6.1.1. LAN (Local Area Network)

LAN digunakan untuk menghubungkan komputer yang berada di dalam suatu area yang kecil, misalnya di dalam suatu gedung perkantoran atau kampus. Jarak antar komputer yang dihubungkannya bisa mencapai 5 sampai 10 km. Suatu LAN biasanya bekerja pada kecepatan mulai 10 Mbps sampai 100 Mbps. LAN menjadi populer karena memungkinkan banyak pengguna untuk memakai sumber daya secara bersama-sama. Contoh dari sumber daya yang dapat digunakan itu misalnya suatu mainframe, *file server*, printer, dan sebagainya.

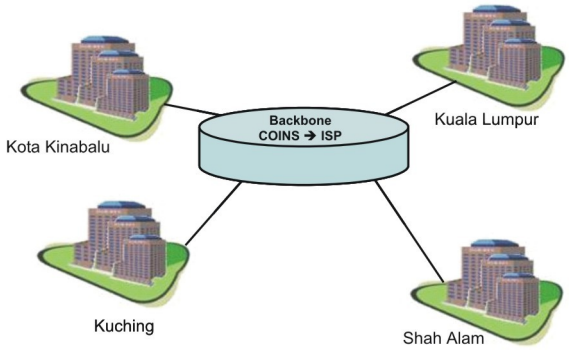


Gambar 2.7. Ilustrasi LAN

2.6.1.2. **MAN (Metropolitan Area Network)**

MAN merupakan suatu jaringan yang cakupannya meliputi suatu kota. MAN menghubungkan LAN-LAN yang lokasinya berjauhan. Jangkauan

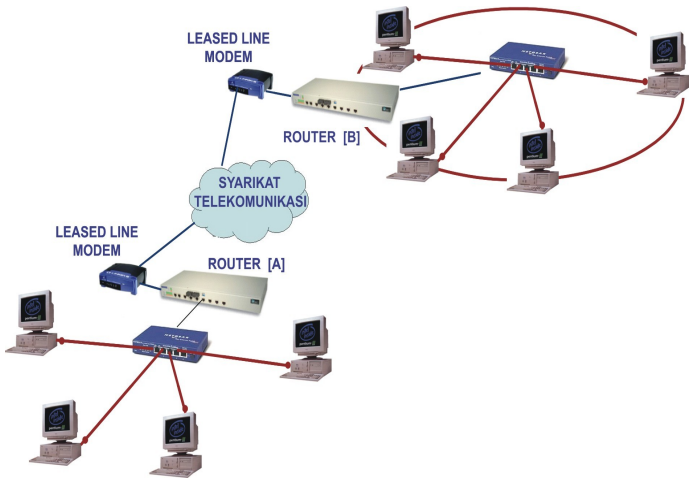
MAN bisa mencapai 10 km sampai beberapa ratus km. Suatu MAN biasanya bekerja pada kecepatan 1,5 sampai 150 Mbps. Pada gambar 2.3 anda dapat melihat suatu ilustrasi tentang MAN.



Gambar 2.8. Ilustrasi MAN

2.6.1.3. WAN (Wide Area Network)

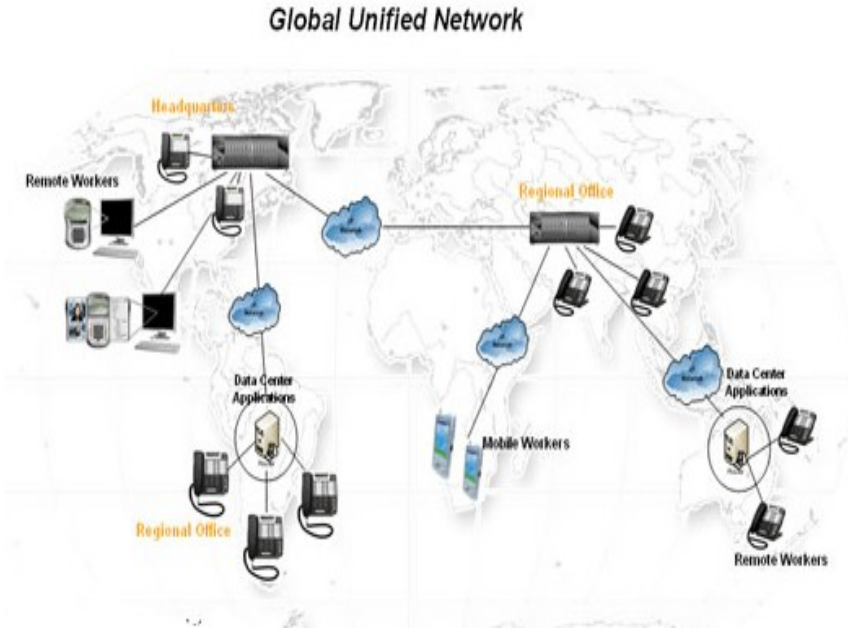
WAN dirancang untuk menghubungkan komputer-komputer yang terletak pada suatu cakupan geografis yang luas, seperti hubungan dari satu kota ke kota lain di dalam suatu negara. Cakupan WAN bisa meliputi 100 km sampai 1.000 km, dan kecepatan antar kota bisa bervariasi antara 1,5 Mbps sampai 2,4 Gbps. Dalam WAN, biaya untuk peralatan transmisi sangat tinggi, dan biasanya jaringan WAN dimiliki dan dioperasikan sebagai suatu jaringan publik. Para pelaku bisnis dapat menyewa sistem transmisi tersebut untuk menghubungkan kantor-kantor cabang yang dimilikinya. Gambar 2.4 mengilustrasikan suatu WAN.



Gambar 2.9. Ilustrasi WAN

2.6.1.4. GAN (Global Area Network)

GAN merupakan suatu jaringan yang menghubungkan negara-negara di seluruh dunia. Kecepatan GAN bervariasi mulai dari 1,5 Mbps sampai dengan 100 Gbps dan cakupannya mencakapi ribuan kilometer. Contoh yang sangat baik dari GAN ini adalah *Internet*. Gambar 2.5 memperlihatkan contoh suatu GAN.



Gambar 2.10.Ilustrasi GAN

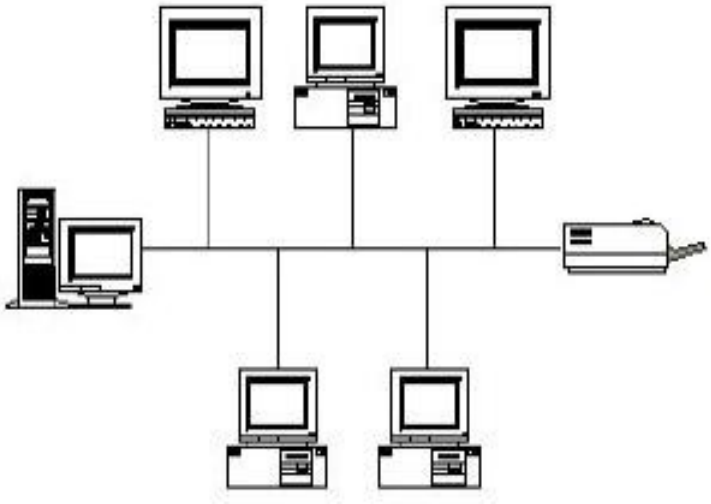
2.6.2. Topologi Jaringan Komputer

Suatu cara menghubungkan komputer yang satu dengan yang lainnya sehingga membentuk jaringan. Cara yang saat ini banyak digunakan adalah *bus, token-ring, star* dan *peer-to-peer network.* Masing-masing topologi ini mempunyai ciri khas, dengan kelebihan dan kekurangannya masing-masing.

1. Topologi *Bus*
2. Topologi *Token ring*
3. Topologi *star*

2.6.2.1. Topologi Bus

Menggunakan satu segment (panjang kabel) *backbone,* yaitu yang menyambungkan semua host secara langsung.



Gambar 2.11. Ilustrasi Topologi Jaringan Bus

Keuntungan menggunakan topologi bus:

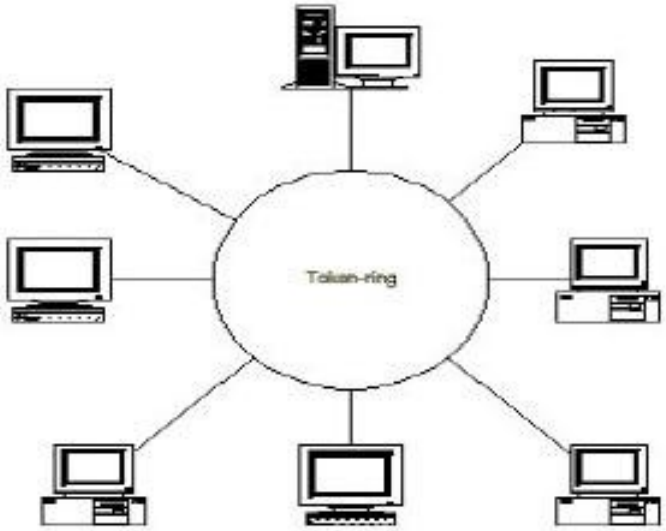
1. Hemat Kabel
2. Layout Kabel Sederhana
3. Mudah dikembangkan

Kerugian menggunakan topologi bus:

1. Deteksi dan isolasi kesalahan sangat kecil
2. kepadatan lalu lintas data
3. Bila salah satu client rusak, maka jaringan tidak akan berfungsi
4. Diperlukan repeater untuk jarak jauh

2.6.2.2. Topologi Token ring

Metode *token ring* (sering disebut juga ring) adalah cara menghubungkan komputer sehingga berbetuk ring (lingkaran). Setiap simpul mempunyai tingkatan yang sama. Jaringan akan disebut loop, data dikirimkan kesetiap simpul dan setiap informasi yang diterima simpul diperiksa alamatnya apakah data itu untuknya atau bukan.



Gambar 2.12. Ilustrasi Topologi Token ring

Keuntungan menggunakan topologi *token ring*:

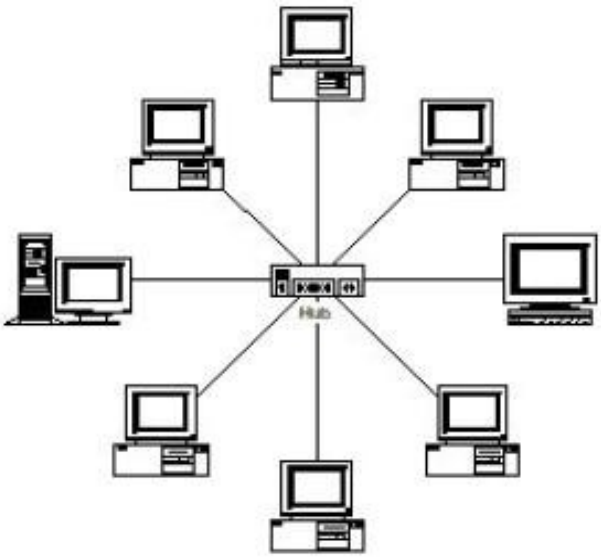
1. Hemat kabel

Kerugian menggunakan topologi *token ring*:

1. Peka terhadap kesalahan
2. Pengembangan jaringan lebih beku

2.6.2.3. **Topologi Star**

Merupakan kontrol terpusat, semua link harus melewati pusat yang menyalurkan data tersebut kesemua simpul atau client yang dipilihnya. Simpul pusat dinamakan stasium primer atau server dan lainnya dinamakan stasiun sekunder atau client server. Setelah hubungan jaringan dimulai oleh server maka setiap client server sewaktu-waktu dapat menggunakan hubungan jaringan tersebut tanpa menunggu perintah dari server. Terdapat keuntungan dan kerugian dari tipe ini yaitu:



Gambar 2.13.Ilustrasi Topologi Star

Keuntungan:

1. Paling fleksibel
2. Pemasangan/perubahan stasiun sangat mudah dan tidak mengganggu bagian jaringan lain
3. Kontrol terpusat
4. Kemudahan deteksi dan isolasi kesalahan/kerusakan
5. Kemudahaan pengelolaan jaringan

Kerugian:

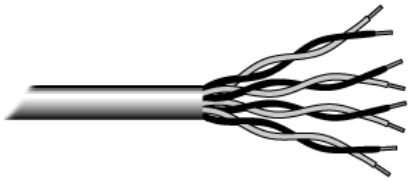
1. Boros kabel
2. Perlu penanganan khusus
3. Kontrol terpusat (HUB) jadi elemen kritis

2.6.3. Media Transmisi Jaringan

Kabel adalah alat penghubung untuk mengirim informasi dari satu komputer ke komputer yang lain. Ada beberapa macam tipe kabel yang umumnya di gunakan bersama LAN. Dalam beberapa kasus, sebuah jaringan hanya menggunakan satu macam tipe kabel, di jaringan yang lain menggunakan

kabel yang berbeda pula. Kabel yang dipilih adalah berdasarkan dengan topologi jaringan, protokol jaringan, dan ukurannya. Ini sangat penting untuk diketahui karena kesuksesan jaringan bergantung dari ke semua aspek tersebut. Salah satunya adalah kabel Twisted Pair, karena kabel ini paling sering digunakan.

Kabel twisted pair ada dua tipe yaitu shielded and unshielded. Unshielded twisted pair (UTP) adalah yang paling populer dan umumnya merupakan pilihan yang terbaik untuk jaringan sederhana. Kualitas kabel UTP berbeda dengan telepon, Kabel ini mempunyai 4 pasang kabel di dalamnya. Setiap pasangan adalah kembar. Standar konektor untuk kabel jenis ini adalah konektor RJ-45.



Gambar 2.14. *Kabel Unshielded twisted pair*



Gambar 2.15. Konektor RG 45

Kabel Shielded Twisted Pair (STP), sangat sensitif terhadap sinyal radio dan listrik. Kabel ini baik sangat baik digunakan dimana lingkungan pengaruh listrik kurang, kabel ini biasanya digunakan pada jaringan yang menggunakan topologi token ring.

2.7 OpenMosix

2.7.1. Sejarah

Gagasan openMosix berawal dari proyek Mosix yang juga menyediakan solusi *clustering* yang transparan di Linux. Proyek Mosix yang dipimpin oleh Profesor Amnon barak dari Hebrew University of Jerusalem diawali pada tahun

1977. Proyek didanai oleh DARPA dan ditahun 1980-an dipakai oleh US Air Force untuk *cluster* pada komputer PDP11/45. Pada 1997 Mosix berjalan pada BSD/OS 3.0 GNU/Linux yang dipilih sebagai platform pengembangan pada tahun 1999.

Pada tahun 2001, proyek Mosix berkembang menjadi versi non-GPL (dari sebelumnya GPL). Oleh karena itu, Mosix tidak lagi bisa dipakai secara bebas. Dr. Moshe Bar, pengajar Tel Aviv University dan UN Atomic Agency, salah seorang yang pernah ikut mengembangkan Mosix, tidak setuju adanya perubahan lisensi tersebut, maka ia, yang merupakan peneliti pada Italian Natioan Institute fir Physical of Matter, dengan perusahaannya (Qluster, Inc) mengawali proyek openMosix untuk menjamin adanya versi gratis yang bisa dipakai secara bebas. openMosix saat ini tidak memuat kode non-GPL(semacam Mosix). Dengan cepat, openMosix meraih popularitas pada SourceForge mulai akhir 2001 dan awal 2002.

Sejak itu, sejumlah besar user Mosix beralih ke openMosix. Saat ini ada 14 orang yang ikut mengembangkan openMosix (sebagai perbandingan, proyek Mosix sendiri hanya dikembangkan oleh 4 orang).

2.7.2. Pengertian

OpenMosix adalah suatu *extension* (ekstensi) pada kernel Linux untuk membentuk *Single-sistem Image* (SSI) *clustering*. Ekstensi pada kernel ini membuat sejumlah jaringan yang terdiri dari PC biasa menjadi semacam *supercomputer* bagi aplikasi Linux.

Bila openMosix telah di-*install* pada beberapa komputer yang terkoneksi pada jaringan yang sama maka *node-node* (istilah PC yang menjadi anggota *cluster*) dalam *cluster* tersebut akan terhubung satu sama lain dan beradaptasi beban kerja mereka (*load balancing*). Proses yang berasal dari suatu *node*, bila berpindah (migrasi) ke *node* lain. openMosix akan bekerja secara kontinyu berusaha mengoptimalkan alokasi sumber daya.

2.7.3. Pola Kerja

Keuntungan utama dari openMosix adalah pembuatan *cluster* yang terdiri baynyak *node* dengan PC yang murah (dibanding membeli SMP sunguhan). OpenMosix seperti halnya sistem SMP, tidak bisa mengeksekusi

suatu proses tunggal pada sejumlah CPU fisik pada saat yang sama. openMosix tidak bisa mempercepat proses tunggal semacam Mozilla, tentu saja, selain memindahkannya ke *node* yang mampu mengeksekusinya lebih efisien. openMosix belum mendukung pemisahan sejumlah *thread* yang saling bekerja sama.

OpenMosix bisa mengubah sejumlah mesin Linux menjadi semacam *virtual SMP* (sebuah arsitektur komputer multiprosesor yang memiliki dua atau lebih prosesor identik dan terhubung ke *shared main memory* tunggal), yang besar. Tentu saja ada perbedaannya. Pada SMP sesungguhnya, dua atau lebih CPU bisa bertukar data dengan sangat cepat, sedang pada openMosix, kecepatan komunikasi *node* ditentukan oleh kecepatan LAN. Pengguna Gigabit Ethernet atau teknologi jaringan berkecepatan tinggi akan memungkinkan efektifitas *cluster* openMosix.

OpenMosix akan memungkinkan eksekusi paralel dengan skalabilitas tinggi pada tingkatan proses. OpenMosix bisa memindahkan kebanyakan proses Linux antar *node*, tanpa mas alah. Jika suatu aplikasi melakukan "fork" proses *child*, masing-masing akan melakukan pekerjaannya sendiri, openMosix bisa melakukan migrasi setiap proses ini ke *node* yang tepat pada *cluster*. Kita bisa mengambil keuntungan dari kemampuan ini meskipun suatu aplikasi tidak bisa bermigrasi secara independen. Sebagai contoh, jika melakukan kompresi 12 *track digital audio* dengan *cluster*, kita bisa memperoleh 12 proses *encoding* yang berjalan simultan. Kemudian openMosix akan memindahkan setiap proses ke *node* yang sesuai pada *cluster*. Jika mempunyai 12 *node*, maka pekerjaan *encoding* akan berjalan 12 kali lebih cepat.

2.7.4. Load Balancing

Salah satu aspek yang penting dalam komputer *cluster* adalah load balancing, suatu mekanisme dimana load / kerja harus dibagi antar komputer, sehingga load / kerja menjadi seimbang. Dengan diterapkannya load balancing pada komputer *cluster*, diharapkan agar sistem komputer *cluster* memiliki kinerja yang optimal.

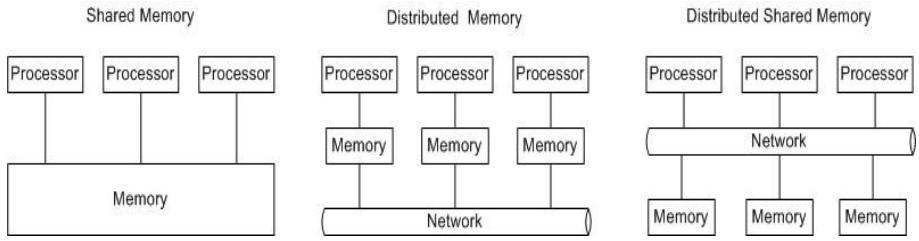
Terdapat beberapa teknik load balancing untuk computer *cluster*. Pendekatan yang digunakan untuk setiap teknik load balancing tersebut juga berbeda-beda. Pendekatan yang umum dan sudah diterima secara luas sebagai

contoh adalah pendekatan dari sisi memory, sisi load / kerja dan sisi penyimpanan (storage). Diperlukan suatu mekanisme penjadwalan yang baik agar sebuah teknik load balancing dapat bekerja secara optimal, dan mekanisme ini biasanya diimplementasikan pada level sistem operasi.

Beberapa contoh teknik load balancing dan pendekatannya :

1. Memory

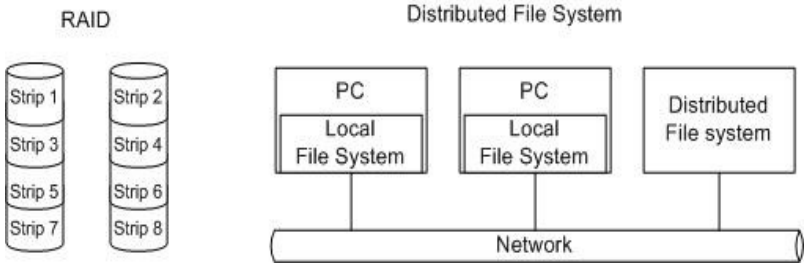
   Pendekatan dari sisi memory ini menekankan pada pengaturan kerja memory pada computer *cluster*, sehingga kinerja sistem computer kluster dapat optimal. Teknik yang biasa digunakan adalah dengan Shared Memory, Distributed Memory dan Distributed Shared Memory.



Gambar 2.16. Shared Memory, Distributed Memory dan Distributed Shared Memory.

2. Storage

   Pendekatan ini menekankan pada kinerja dari media penyimpanan (*storage*). Pendekatan ini telah diimplementasikan dalam berbagai teknik, seperti RAID (*Redundant Array of Independent Disks)* dan DFS (*Distributed File system*)



Gambar 2.17. RAID dan DFS

3. Load

   Pendekatan dari sisi load menekankan pada pengaturan kerja/load dari setiap komputer. Load / kerja yang dimaksud adalah proses yang harus dieksekusi. Teknik paling tua yang sampai sekarang masih digunakan adalah metode round robin beserta variasinya. Teknik lainnya adalah dengan mekanisme migrasi proses**.**

Load balancing merupakan hal penting untuk kinerja pemrosesan paralel. Ide dasarnya adalah memindahkan proses dari *node* yang memilih beban kerja tinggi ke *node* yang memiliki beban kerja rendah. Tujuannya agar waktu pengerjaan tugas lebih rendah dan menaikkan utilitas prosesor.

Bagian penting dari strategi dari *load balancing* adalah *migration policy*, yang menentukan kapan suatu migrasi terjadi dan proses mana yang dimigrasikan. Proses migrasi pada *load balancing* terjadi dua bentuk, yaitu :

1. *Remote execution* (juga disebut *non-preemptive migration*). Pada strategi ini proses baru (bisa secara otomatis) dieksekusi pada *host remote*.
2. *Pre-emptive migration*, pada strategi ini proses akan dihentikan, dipindahkan ke *node* lain, dan diteruskan.

Algoritma pada openMosix bersifat desentralisasi, maka suatu *node* dapat ditambahkan atau dikurangi dari konfigurasi *cluster* tanpa menimbulkan gangguan. OpenMosix memiliki *cluster* tanpa menimbulkan gangguan. openMosix memiliki kemampuan *self tuning* untuk mendeteksi kecepatan tiap *node*, memonitor beban sisa memory, dan laju IPC dan I/O, openMosix memakai *pre-emptive migration* dengan *load balancing* otomatis, tetapi *user* juga bisa melakukan intervensi penentuan secara eksplisit.

User context di sebut *remote*, memuat kode program, *stack*, *memory-map*, dan registrasi dari proses. *Remote* mengenkapsulasi proses saat berjalan dalam tingkatan *user* (*user level*).

2.7.5. Migrasi Proses

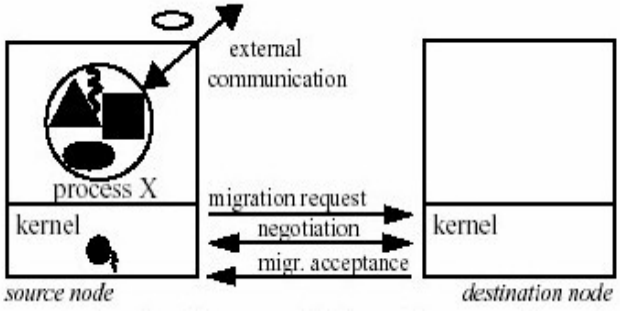
Faktor terpenting dalam suatu *cluster* adalah adanya migrasi proses dari *node* satu ke *node* lain. *User* bisa mengawali pekerjaannya dengan menjalankan sejumlah proses pada satu *node*. Kemudian, sistem akan memindahkan proses ke *node* terbaik yang ada. Jika selama eksekusi ada sumber daya akan memakainya dengan melakukan penugasan (*assign*) ulang. Kemampuan untuk "*assign* dan *reassign*" ini merupakan hal yang terpenting untuk kemudahan penggunaan dan memungkinkan lingkungan eksekusi *time-sharing* yang efisien.

OpenMosix mendukung migrasi proses yang *preemptive* dan sepenuhnya transparan. Setelah migrasi, suatu proses terus berinteraksi dengan lingkungannya tidak bergantung pada lokasinya. Untuk menerapkan PPM (*Preemptive* migrasi proses), proses yang bermigrasi dibagi dalam dua konteks, yaitu:

1. *User context* yang bisa bermigrasi
2. *System context* bergantung pada UHN (*Unique Home-node*) dan tidak bisa berpindah.

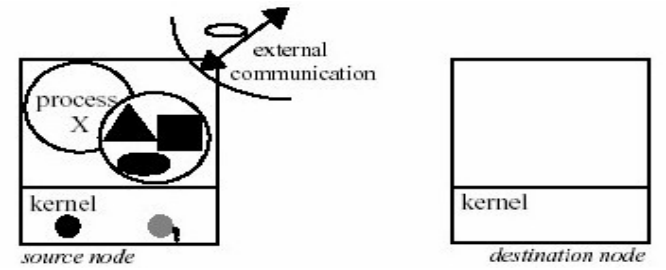
2.7.5.1. Algoritma Migrasi Proses

Walaupun ada bermacam-macam design dan implementasi dari migrasi proses, namun pada umumnya mempunyai algoritma seperti terlihat pada gambar



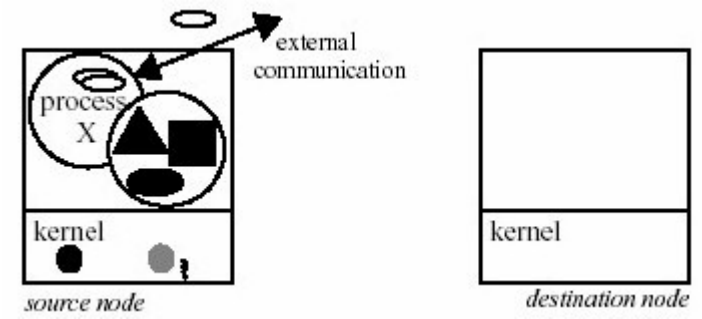
Gambar 2.18.Permintaan Proses

Permintaan untuk melakukan migrasi dilakukan oleh sebuah komputer kepada komputer lain, setelah melakukan negosiasi, permintaan migrasi diterima.



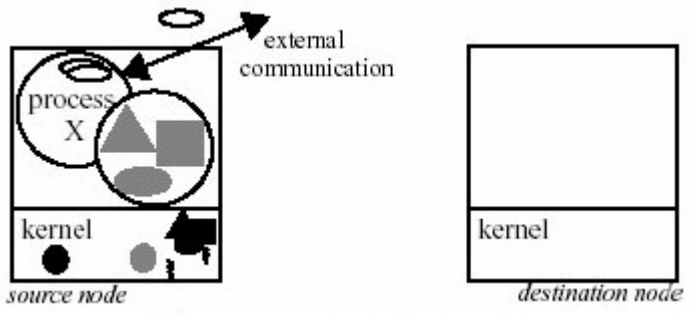
Gambar 2.19. Pelepasan Proses

Sebuah proses dilepaskan dari komputer asalnya dengan menunda eksekusi proses tersebut, menyatakan bahwa proses tersebut sedang dalam proses melakukan migrasi dan melakukan mekanisme penyimpanan dan meneruskan semua komunikasi yang terjadi.



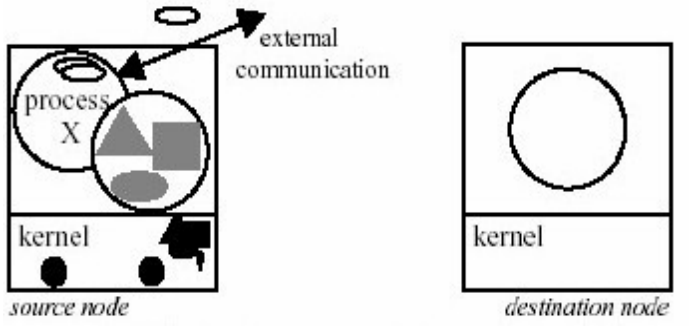
Gambar 2.20. Komunikasi Diteruskan

Untuk sementara komunikasi diteruskan (*redirecting*) dengan cara menampung semua komunikasi yang ditujukan untuk proses yang sedang melakukan migrasi, dan kemudian meneruskannya kepada proses yang melakukan migrasi seteleh migrasi proses selesai. Proses redirecting ini berlangsung secara parallel sampai komunikasi diaktifkan kembali. Setelah komunikasi diaktifkan kembali, maka semua komputer dalam *cluster* akan mengetahui proses tersebut telah melakukan migrasi, sehingga seluruh komunikasi untuk proses tersebut akan langsung ditujukan kepada komputer tempat proses tersebut berada.



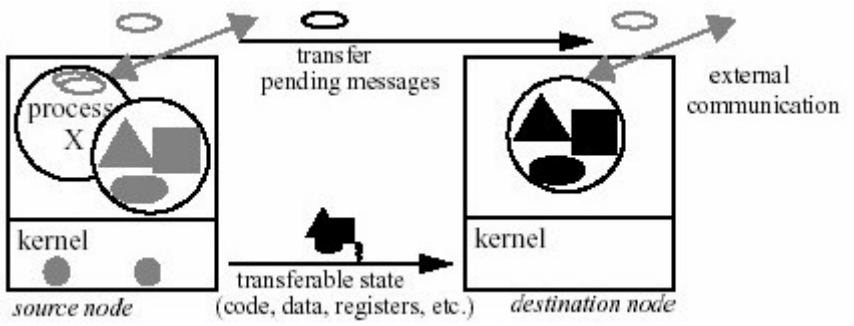
Gambar 2.21. Pengambilan State Proses

State dari proses diambil, state ini antara lain adalah : isi dari memory, state dari prosesor (register), state dari komunikasi (file yang dibuka). State dari proses akan berada terus pada komputer asal selama migrasi berlangsung. Setelah migrasi proses selesai, state tersebut akan dihapus. Namun pada beberapa sistem, state tersebut akan terus tersimpan dalam komputer asal.



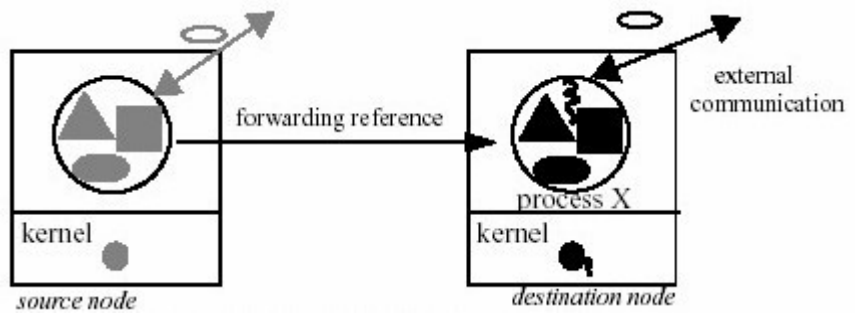
Gambar 2.22. Pembuatan Proses pada Komputer Tujuan

Sebuah proses dibuat pada komputer tujuan, dimana seluruh state akan ditransfer kepada proses baru ini. Proses baru ini tidak akan aktif sampai seluruh state yang ditransfer selesai diimplementasikan pada proses tersebut.



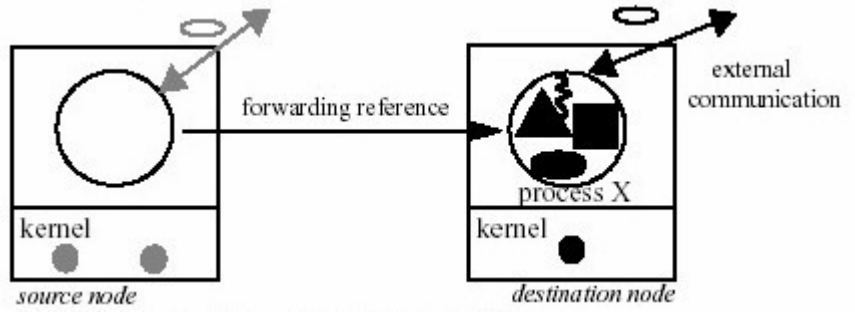
Gambar 2.23. Implementasi State pada Proses Tujuan

State ditransfer dan diimplementasikan pada proses baru yang terbentuk pada komputer tujuan. Proses baru tersebut telah aktif dan menjadi proses regular pada komputer tujuan.



Gambar 2.24. Pengaktifan Kembali Komunikasi

Seluruh komunikasi yang terhenti dan ditampung dalam antrian pada komputer asal diteruskan kepada komputer tujuan, dan seluruh komputer pada *cluster* mengetahui bahwa proses tersebut telah melakukan migrasi.



Gambar 2.25. Proses dilanjutkan

Proses dilanjutkan pada komputer tujuan. migrasi proses telah selesai dan proses pada komputer asal dapat dihapus.

2.7.6. Sistem File

Bagaimana proses yang bermigrasi memperoleh akses sistem file Bila suatu proses membuat *temporary file* pada suatu *node*, maka proses tersebut harus tetap memelihara akses ke file tadi, meski telah berpindah ke *node* lain. Terdapat tiga macam sistem file yang perlu diketahui yaitu:

1. DFSA (*Direct File system Access*), merupakan suatu teknik optimasi. Ini memungkinkan remote process untuk melakukan sistem call pada file secara local daripada mengirimkannya ke home *node*.
2. MFS (*Mosix File system*) memungkinkan semua *node* mengakses semua sistem file *node*. DFSA berjalan pada suatu sistem file *cluster*, pada kasus ini MFS. Ini dipakai oleh Mosix, tidak oleh openMosix.
3. oMFS (*openMosix File system*) adalah suatu sistem file yang dipakai oleh kernel openMosix. oMFS dapat diperoleh dengan cara menginstall suatu kernel openMosix pada *node* di *cluster* dengan mengaktifkan oMFS pada kernel-config (secara *default* biasanya sudah aktif saat format *installer* RPM). OpenMosix bisa memakai oMFS dan DFSA. Keduanya, akan memberikan sejumlah fitur yang membuat akses ke sistem *file* lebih transparan bagi *user.*

2.8 Disain Penelitian

Tujuan yang lebih luas dari suatu disain penelitian (*design of experiment*) adalahuntuk meningkatkan informasi yang diperoleh. Karena itu maka informasi yang tepat dan benar dapat diharapkan jika tidak terdapat banyak kekeliruan sewaktu mengadakan pengukuran, pemilihan subjek, analisis data, dan kegiatan lainnya selama prosuder penelitian dijalankan.

2.8.1. Metode Faktorial 2 Faktor

Pengukuran dan analisis data yang digunakan pada penelitian ini adalah metode desain faktorial 2 faktor (2^k *factorial design*), dengan faktor 2 kali 2 (2x2). Dua Variabel pertama merupakan dua jenis CPU—PIII 400 Mhz dan AMD 1500+, dan variabel yang kedua adalah dua jenis ukuran RAM yang digunakan pada masing-masing jenis CPU—64 MB-128 MB, dan 128 MB-256 MB.

1. 2^2 *factorial design*

Pada masing-masing faktor memiliki dua tingkatan atau bagian, misalnya pada Tabel 2.1:

Tabel 2.1. *Contoh 2^2 factorial design*

Total Waktu Eksekusi Pada CPU/1500 (s)		
Jumlah PC	Jumlah RAM	
	128 MB	256 MB
2	2214,784000	1229,120000
4	814,360000	625,216000

dimana, X_A= -1 jika memori 128MB

1 jika memori 256MB

X_B= -1 jika jumlah PC 2 *node*

1 jika jumlah PC 4 *node*

2. Mode dari 2^2 desain faktorial

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B \quad \text{..... (2.1)}$$

Contoh:

Tabel 2.2. *Contoh eksperimen 2^2 factorial design*

Eksperimen	A	B	Y
1	-1	-1	y_1
2	1	-1	y_2
3	-1	1	y_3
4	1	1	y_4

dengan model :

$$y = q_0 + q_A x_A + q_B x_B + q_{AB} x_A x_B \quad \text{..... (2.2)}$$

Subtitusi:

$$y_1 = q_0 - q_A - q_B + q_{AB} \quad \text{..... (2.3)}$$

$$y_2 = q_0 + q_A - q_B - q_{AB} \quad \text{..... (2.4)}$$

$$y_3 = q_0 - q_A + q_B - q_{AB} \quad \text{..... (2.5)}$$

$$y_4 = q_0 + q_A + q_B + q_{AB} \quad \text{..... (2.6)}$$

3. Solusi

$$q_1 = \frac{1}{4}(y_1 + y_2 + Y_3 + Y_4) \quad \text{..... (2.7)}$$

$$q_A = \frac{1}{4}(-Y_1 + Y_2 - Y_3 + Y_4)$$ ..... (2.8)

$$q_B = \frac{1}{4}(-Y_1 - Y_2 + Y_3 + Y_4)$$ ..... (2.9)

$$q_{AB} = \frac{1}{4}(Y_1 - Y_2 - Y_3 + Y_4)$$ ..... (2.10)

dimana, q_A = kolom A x kolom y

q_B= kolom B x kolom y

q_{AB}= kolom A x kolom B x kolom y

4. Variasi karena kurva A = SSA=$2^2 x q_A$ ..... (2.11)
   Variasi karena kurva B = SSB=$2^2 x q_B$ ..... (2.12)
   Variasi karena kurva AB = SSAB=q_{AB}^2 ..... (2.13)
5. Sum square total (SST)=SSA+SSb=SSAB ..... (2.14)
6. Persentase kontribusi penambahan clock CPU terhadap peubah kecepatan $\frac{SSA}{SST} x100\%$ ..... (2.15)

   Persentase kontribusi penambahan jumlah *node* terhadap peubah kecepatan $\frac{SSB}{SST} x100\%$ ..... (2.16)