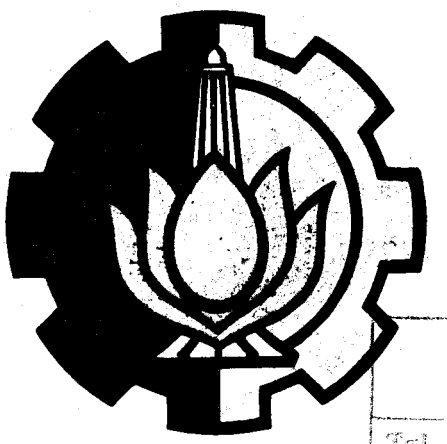
PENGEMBANGAN SISTEM BACKUP PADA PARALEL VIRTUAL FILE SYSTEM DI LINGKUNGAN SISTEM OPERASI LINUX

TUGAS AKHIR



Oleh :

ROYYANA MUSLIM IJTIHADIE
2695100001

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2000

PENGEMBANGAN SISTEM BACKUP PADA PARALEL VIRTUAL FILE SYSTEM DI LINGKUNGAN SISTEM OPERASI LINUX

TUGAS AKHIR

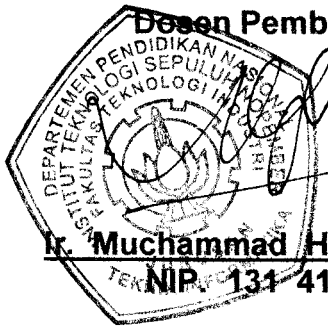
Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer

Pada

Jurusan Teknik Informatika
Fakultas Teknologi Sepuluh Nopember
S U R A B A Y A

Mengatahui / Menyetujui

Dosen Pembimbing



Ir. Muchammad Husni, M.kom
NIP. 131 411 100

S U R A B A Y A
Agustus, 2000

ABSTRAK

Seiring dengan meningkatnya popularitas paralel prosesing, dibutuhkan pula sebuah resource yang murah untuk paralel computing. Kumpulan workstation merupakan salah satu cara untuk menyediakan fasilitas paralel komputer dengan harga yang murah daripada paralel komputer yang sebenarnya. Kumpulan komputer yang digunakan sebagai penyedia komputasi pararel ini dapat dibangun dari workstation (PC) yang tersambung pada jaringan. File system yang terdistribusi merupakan salah satu cara untuk memanfaatkan resource dari kumpulan workstation tersebut. Sehingga resource disk yang dimiliki masing masing workstation tersebut bisa diakses secara bersama sama. Paralel Virtual File System merupakan salah satu alternatif untuk melakukan hal tersebut. Dengan adanya file system ini diharapkan bisa didapatkan ruang disk yang lebih besar karena menggunakan banyak sumber daya (resource) disk.

KATA PENGANTAR

Bismillahirrahmanirrahim

Dengan nama Allah yang Maha Pengasih dan Maha Penyayang. Sholawat dan salam bagi Nabi Muhammad SAW. Segala Puji bagi Allah SWT yang telah memberikan rahmat-Nya sehingga penulis dapat menyelesaikan tugas akhir dengan judul :

" Implementasi Sistem Backup Pada Paralel Virtual File System di Lingkungan Sistem Operasi Linux "

Tugas akhir ini disusun guna memenuhi salah satu persyaratan untuk menyelesaikan perkuliahan di Teknik Informatika ITS serta memperoleh gelar sarjana. Penulis berharap tugas akhir ini dapat memberikan manfaat dan tambahan pengetahuan bagi semua pihak.

Penyusunan tugas akhir ini tidak lepas dari bantuan banyak pihak. Oleh karena itu pada kesempatan kali ini penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada :

1. Bapak M. Achmad Kafrawi dan Ibu Innaha Setyawati, orangtua penulis yang sangat penulis sayangi, yang telah memberikan semangat, doa dan segalanya bagi penulis yang tidak dapat penulis ungkapkan.
2. Bapak Ir. M. Husni, M.Kom, sebagai dosen pembimbing. Yang telah memberikan bimbingan dan dukungan pada penulis terutama selama pengerjaan tugas akhir
3. Bapak Ir. Arif Djunaedy sebagai Ketua Jurusan Teknik Informatika ITS.

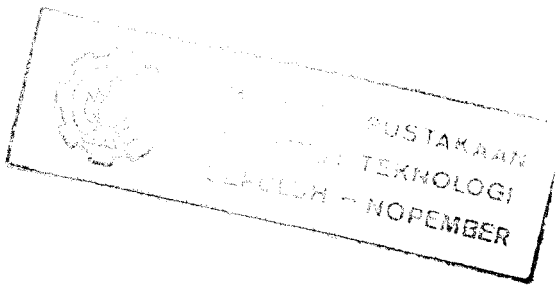
4. Bapak Suhadi Lili sebagai dosen pembimbing.
5. Bapak dan Ibu Dosen di Jurusan Teknik Informatika ITS yang telah memberikan ilmunya selama penulis duduk di bangku kuliah TC
6. Tanti, Riris dan Rima, saudara-saudara penulis, yang amat penulis sayangi.
7. Dini, yang banyak memberikan dukungan, dorongan dan semangat yang tak ternilai bagi penulis
8. Heri 'G', Pramudyo, Gandi, Wawan 'M', Jayeng, Hartono, Fauzi, cak Yul, Salim, Imam KD, Rozak, Ade, Bornok, Pande, Pak Toha, Hermanu, Feri, dan semua teman-teman C0b (cowok) atas kerjasamanya selama kuliah di TC.
9. Eni, Yuliana, Naim, Elok dan semua teman C0b (putri) atas kerjasamanya selama kuliah di TC.
10. Pak Dwi dan Tim TC-Net yang telah memberikan fasilitas selama pengerjaan TA.
11. Tim ITS-net ITS sekarang dan yang lalu , mbak Muji, Mbak Yenni, Mas Cahya, Dotok, Kilon, widianto, pakde Inu, bang joko dan kawan-kawan yang telah memberikan kelonggaran bagi penulis selama pengerjaan TA.
12. Pak Gunarta, Pak Salty, Komang, Pande, Andreas, Mnason dan teman-teman lain di MKI yang telah memberikan dorongan serta kelonggaran selama pengerjan TA.
13. Danil, 'tetangga' dalam mengerjakan TA di Lab. Pemrograman.
14. Kendy, Rizky, Dimas, Dian, Ajun, Nur, Suyudi, Dhias, Hera, Daning, Wahyudi yang setia 'menunggu' lab dan menemani penulis.
15. Tim sepakbola TC, Gershom, Mario, Bang Darwan, Sonny, Widi dan kawan-kawan.

16. Asfik, Hudan, Hermono, Pak Soleh, Mas Sugeng, Pak Kodir, Mas Yudi dan staff karyawan Teknik Informatika yang telah banyak membantu dalam segala hal.
17. Pak Mu'in, Pak Karmono, Pat Riatmoko dan Guruh yang telah membantu menjaga keamanan dan menjadi teman ngobrol.
18. Pak Nengah dan Ibu Kusumastuti selaku orangtua kos sejak awal kuliah
19. Semua pihak yang tidak dapat disebutkan satu persatu yang ikut membantu penulis selama ini.

Semoga Allah SWT memberikan rahmat dan hidayah-Nya kepada kita semua

Amin.

Surabaya, Agustus 2000



DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

PENDAHULUAN

1.1 LATAR BELAKANG

Seiring dengan meningkatnya popularitas paralel prosesing, dibutuhkan pula sebuah resource yang murah untuk paralel computing. Kumpulan workstation merupakan salah satu cara untuk menyediakan fasilitas paralel komputer dengan harga yang murah daripada paralel komputer yang sebenarnya. Kumpulan komputer yang digunakan sebagai penyedia komputasi pararel ini dapat dibangun dari workstation (PC) yang tersambung pada jaringan.

Tiap mesin mempunyai komposisi hardware yang berbeda-beda tergantung dari sarana yang tersedia. Mesin-mesin tersebut menggunakan software khusus untuk menyeimbangkan kerja antar mesin. Penggunaan PC dimaksudkan untuk lebih menekankan penggunaan paralel prosesing dengan harga murah dengan memanfaatkan hardware yang umum di pasaran.

Pararel komputer yang dibangun dengan menggunakan kumpulan *workstation* menimbulkan permasalahan diantaranya adalah file system dan interaksi aplikasi, termasuk ketidakcocokan antar susunan fisik data dan penyebaran tugas. Kekurang koordinasian antar tugas aplikasi menghasilkan kemampuan disk yang rendah dan kemampuan akses yang rendah pula. Selain itu paralel komputer tersebut juga membutuhkan

sebuah perangkat lunak untuk menggabungkan *storage* yang terdapat pada masing-masing workstation sehingga terlihat sebagai satu kesatuan tempat penyimpanan (*storage*) bagi user. Paralel file system adalah salah satu cara pendekatan untuk menyediakan akses global dalam system.

Paralel File System adalah software yang didesain untuk mendistribusikan data melalui beberapa resource I/O dalam paralel computer system dan untuk mengatur akses paralel ke data tersebut.

Paralel file system menyediakan dua fungsi utama yaitu meletakkan data yang terletak dalam file tunggal yang secara fisik didistribusikan melalui I/O dan menyediakan mekanisme untuk aplikasi paralel sehingga data bisa diakses secara konkuren.

Paralel Virtual File System (PVFS)merupakan suatu perangkat lunak untuk menyediakan paralel file system untuk PC Cluster. Sebagai paralel file system, PVFS menyediakan kemampuan *striping* data melalui berbagai I/O *Nodes* dan juga berbagai *user interface*. PVFS menggunakan protocol TCP/IP.

Tujuan PVFS adalah menyediakan paralel file system yang didesain untuk menyesuaikan dengan karakteristik Pile-Of-PCs (kumpulan PC) dengan antar muka yang didesain untuk menghubungkan dengan aplikasi yang sesuai dengan *workload* yang ada.

Sistem ini merupakan hasil percobaan yang dilakukan pada kedua teknik tersebut untuk transfer data antar aplikasi dan file system serta merupakan antarmuka untuk interaksi dengan file system. Salah satu kebutuhan dari PVFS adalah membangun PVFS tanpa memodifikasi file system yang sudah ada dalam platform. Untuk alasan ini maka PVFS

dibuat sebagai daemon dan sebuah *library* yang bisa digunakan aplikasi yang menggunakan PVFS, dengan demikian maka PVFS bisa beroperasi dalam berbagai jenis lingkungan software sistem operasi.

1.2 TUJUAN DAN MANFAAT

Tujuan dari tugas akhir ini adalah mengimplementasikan paralel virtual file system sebagai file system alternatif untuk diterapkan pada array disk yang tersebar pada jaringan.

Manfaat dari tugas akhir Pengimplementasian Paralel Virtual File System adalah menyediakan *cluster-wide consistent name space* (pengaksesan direktori dari beberapa komputer dengan satu nama yang sama) dan juga striping data (*user-controlled striping of data*) lintas *node* dan menyediakan akses data berkecepatan tinggi pada operasi file.

1.3 PERUMUSAN MASALAH

Ruang lingkup pembahasan dari Tugas Akhir ini adalah sebagai berikut:

- Pengimplementasian Paralel Virtual File System untuk mendistribusikan data dengan menggunakan clustered workstation.
- Membuat sistem backup untuk jaringan yang menggunakan Paralel Virtual File System

1.4 BATASAN MASALAH

Batasan masalah dari tugas akhir ini adalah sebagai berikut :

- Cluster PC yang digunakan sebagai bagian dari Paralel Virtual File System adalah PC yang up 24 jam nonstop dan diasumsikan tidak pernah down
- Sistem operasi yang digunakan adalah sistem operasi Linux Redhat kernel 2.2.x

1.5 SISTEMATIKA PENULISAN

Sistematika penulisan yang digunakan dalam penyusunan buku tugas akhir ini adalah sebagai berikut :

BAB I PENDAHULUAN

Merupakan pendahuluan yang berisikan latar belakang pembuatan tugas akhir.

BAB II LANDASAN TEORI

Berisikan landasan teori tentang teori dasar dari file system pada system operasi Linux, yang terdiri dari Linux File system, second extended file system, dan modul kernel file system

BAB III PARALEL VIRTUAL FILE SYSTEM

Berisi penjelasan tentang perangkat lunak PVFS secara detil. Meliputi arsitektur, dan pemrosesan request

BAB IV PENGEMBANGAN SISTEM BACKUP PADA PARALEL VIRTUAL FILE SYSTEM

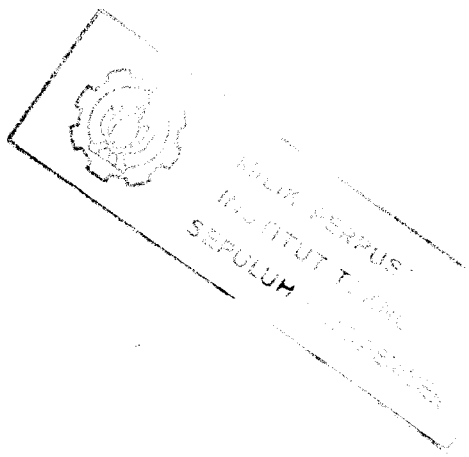
Berisi arsitektur dan perancangan perangkat lunak sistem backup pada PVFS yang diterapkan pada sistem operasi Linux.

BAB V IMPLEMENTASI PVFS

Berisikan implementasi penggunaan PVFS dan hasil percobaan menggunakan PVFS.

BAB VI KESIMPULAN DAN SARAN

Berisikan kesimpulan dan hasil evaluasi serta saran dari pengerjaan tugas akhir.



BAB II

LANDASAN TEORI

Pada bab ini akan dijelaskan mengenai teori dasar dari file system pada system operasi Linux, yang terdiri dari Linux File system, second extended file system, dan modul kernel file system

2.1 LINUX FILE SYSTEM

Pada mulanya Linux merupakan sistem operasi yang diturunkan dari sistem operasi minix. Pada waktu itu lebih mudah membagi disk antara dua sistem daripada membuat file system baru, sehingga Linus Torsvald, pencipta linux memutuskan untuk mengimplementasikan dukungan untuk file system minix di linux. Akan tetapi file system minix terlalu terbatas, sehingga para pengguna kemudian mendevelop file system baru untuk diterapkan di linux. Untuk memudahkan penambahan file system tambahan dalam linux kernel, maka para pengguna akhirnya mendevelop Layer VFS (Virtual File System). VFS ini petama kali dikembangan oleh Chris Provenzano dan akhirnya kemudian diintegrasikan dalam Linux kernel oleh Linus Torsvald.

Setelah intregrasi VFS dalam linux kernel, file system baru tersebut yang disebut dengan Extended File System tersebut diimplementasikan pada April 1992 pada sistem operasi Linux versi 0.96c. File system ini menghilangkan 2 keterbatasan minix. Akan tetapi file system ini masih belum mendukung untuk akses terpisah . File system

menggunakan link list untuk menyimpan jalur dari blok dan inode yang bebas . Hal ini mengakibatkan file system menjadi terfragmentasi.

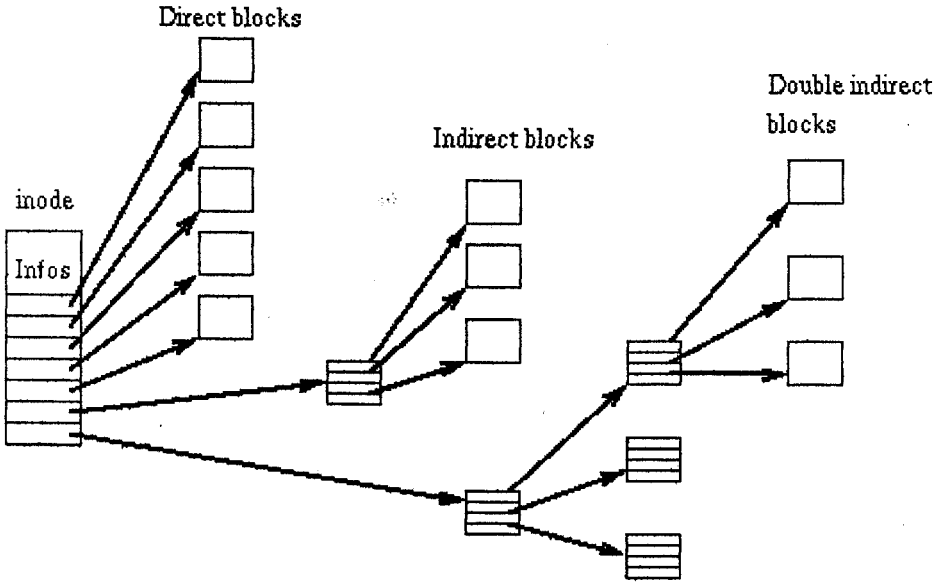
Sebagai response dari hal ini, maka dikembangkan lagi sebuah file system yang merupakan pengembangan dan perbaikan dari extended file system yang disebut dengan Second Extended File system (ext2fs) yang mempunyai kelebihan mendukung nama file panjang, dan mendukung partisi yang lebih besar.

2.2 KONSEP DASAR FILE SYSTEM

Tiap tiap file system linux mengimplementasikan kumpulan dasar dari konsep yang umum yang diturunkan dari sistem operasi Unix. File direpresentasikan oleh inode, direktori dan file yang beirsikan daftar entri dan device yang bisa diakses.

2.2.1 INODES

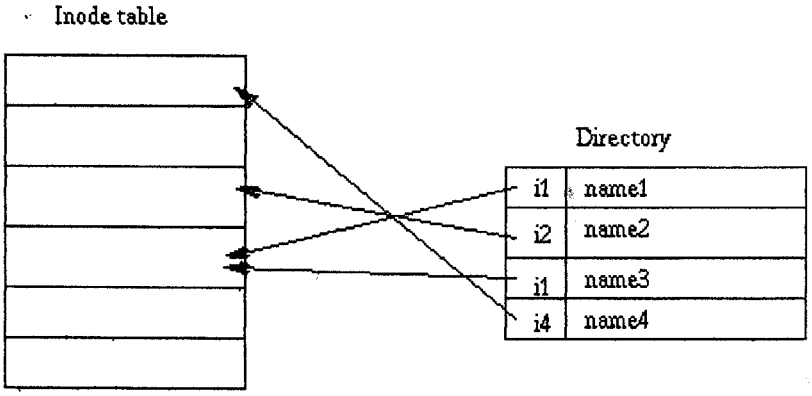
Tiap file direpresentasikan oleh struktur, yang disebut inode. Tiap inode berisikan deskripsi tentang file, yaitu jenis file, hak akses, owner, timestamp, ukuran, dan pointer pada datablok. Alamat data blok dialokasikan pada file yang disimpan pada inode tersebut. Ketika user meminta operasi I/O pada file, kode kernel akan mengkonversi offset yang ada kepada nomor blok, menggunakan nomor ini sebagai index dalam tabel blok alamat dan membaca atau menulis blok fisik. Gambar berikut merepresentasikan struktur inode



Gambar 2.1
Struktur inode Linux File System

2.2.2 DIREKTORI

Direktori disusun dalam bentuk pohon hirarki. Tiap direktori bisa berisikan file file dan subdirektori. Direktori diimplementasikan sebagai sebuah jenis file yang khusu. Sebenarnya sebuah direktori adalah sebuah file yang berisikan daftar entri. Tiap entri berisikan nomor inode dan nama file. Ketika proses menggunakan pathname, maka kode kernel akan mencari dalam direktori untuk menemukan inode number yang bersangkutan. Setelah nama dikonversi pada nomor Inode, maka inode akan dimuat ke memory dan digunakan oleh proses yang meminta. Gambar berikut ini merepresentasikan sebuah direktori.



Gambar 2.2
Representasi direktori dalam linux file system

2.2.3 LINK

File system unix menerapkan apa yang disebut dengan link. Beberapa nama bisa diasosiasikan dengan sebuah inode. Inode akan berisikan field yang berisikan nomor yang berasosiasi dengan file. Menambahkan sebuah link sama dengan membuat sebuah entri direktori yang mana nomor inode menunjuk pada inode.

2.2.4 FILE DEVICE

Dalam sistem operasi unix, device bisa diakses melalui file khusus (special file). Sebuah file khusus device tidak menggunakan space dalam file system. File tersebut merupakan sebuah titik akses menuju ke device driver. Dua jenis file khusus (special file) yang ada yaitu character dan blok. File tersebut direferensikan oleh major number, untuk menentukan jenis device , dan minor number untuk menentukan unit.

2.3 SECOND EXTENDED FILE SYSTEM

Second extended file system yang dikenal dengan ext2fs menerapkan banyak perkembangan pada extended file system. Yang mana banyak kekurangan kekurangan pada extended file system yang dibenahi pada ext2fs ini

Ext2fs menyediakan long file name, yang menggunakan direktori entri. Maximum ukuran nama adalah 255 karakter. Batasan ini bisa ditambah hingga mencapai 1012 jika dibutuhkan. Ext2fs mencadangkan beberapa blok untuk super user (root) . Biasanya 5 persen dari jumlah blok di cadangkan. Hal ini memungkinkan administrator untuk merecover secara mudah.

Ext2fs memungkinkan administrator untuk memilih ukuran dari logical blok ketika membuat file system. Ukuran block bisa jadi 1024,2048 dan 4096. Penggunaan blok ukuran besar bisa mempercepat operasi I/O , karena mengurangi pencarian di harddisk. Akan tetapi ukuran blok yang besar akan menghabiskan disk space.

Ext2fs menerapkan fast symbolic link. Fast symbolic link tidak menggunakan datablok dalam file system. Nama tujuan tidak disimpan dalam data blok tetapi dalam inode itu sendiri. Hal ini bisa menghemat disk space dan mempercepat operasi link. Maximum ukuran nama yang bisa dipakai adalah 60 karakter.

Ext2fs menyimpan jalur dari state file system. Sebuah field khusus dalam superblok digunakan oleh kode kernel untuk mengindikasikan status file system. Ketika file system dimount dalam read/write mode, maka state-nya akan diset pada 'Not clean'. Pada waktu boot file system

checker akan menggunakan informasi ini untuk menentukan file system mana yang harus dicheck.

Mengabaikan file system check bisa jadi berbahaya, jadi ext2fs menyediakan dua jalan untuk memaksakan selang waktu untuk cek regular. File system menyimpan sebuah mount counter dalam super blok yang mana counter ini akan dinaikkan hingga mencapai batas maximal . Jika telah sampai pada batas maximal maka file system checker akan memeriksa apakah file system telah 'clean'.

Struktur fisik dari file system Ext2fs

BOOT SECTOR	BLOCK GROUP 1	BLOCK GROUP 2	BLOCK GROUP N

Gambar 2.3
Struktur fisik file system ext2fs

Tiap-tiap block group berisikan salinan dari backup dari informasi file system (superblok dan deskriptor file system) dan juga berisikan bagian dari file system (bitmap blok, bitmap inodem dan bagian dari tabel inode dan juga blok data). Struktur dari blok group direpresentasikan dalam tabel berikut :

SUPER BLOCK	FS DESCRIPTORS	BLOCK BITMAP	INODE BITMAP	INODE TABLE	DATA BLOCKS

Gambar 2.4
Struktur blok file system ext2fs

Penggunaan block group adalah k, karena control struktur di replikasi dalam tiap block group, maka operasi untuk merecover file system menjadi lebih mudah. Struktur ini juga membantu untuk

mendapatkan performa yang bagus dengan cara mengurangi jarak antara tabel inode dan blok data, sehingga akan mengurangi pula pencarian oleh head dalam file.

Dalam ext2fs direktori diatur sebagai sebuah link list yang berisikan entri variable-length. Tiap entri berisikan nomor inode, panjang entri, nama file dan panjangnya. Dengan menggunakan entri variable-length, maka dimungkinkan untuk mengimplementasikan nama file panjang (long file names) tanpa menghabiskan space harddisk pada direktori. Struktur dari entri direktori adalah sebagai berikut:

INODE NUMBER	ENTRY LENGTH	NAME LENGTH	FILENAME

Gambar 2.5
Struktur entri direktori ext2fs

Sebagai contoh, tabel berikut merepresentasikan struktur dari direktori yang berisikan tiga buah file yang bernama file 1, long_file_name, dan f2:

I1	6	5	File1
I2	0	4	Long_file_name
I3	2	2	F2

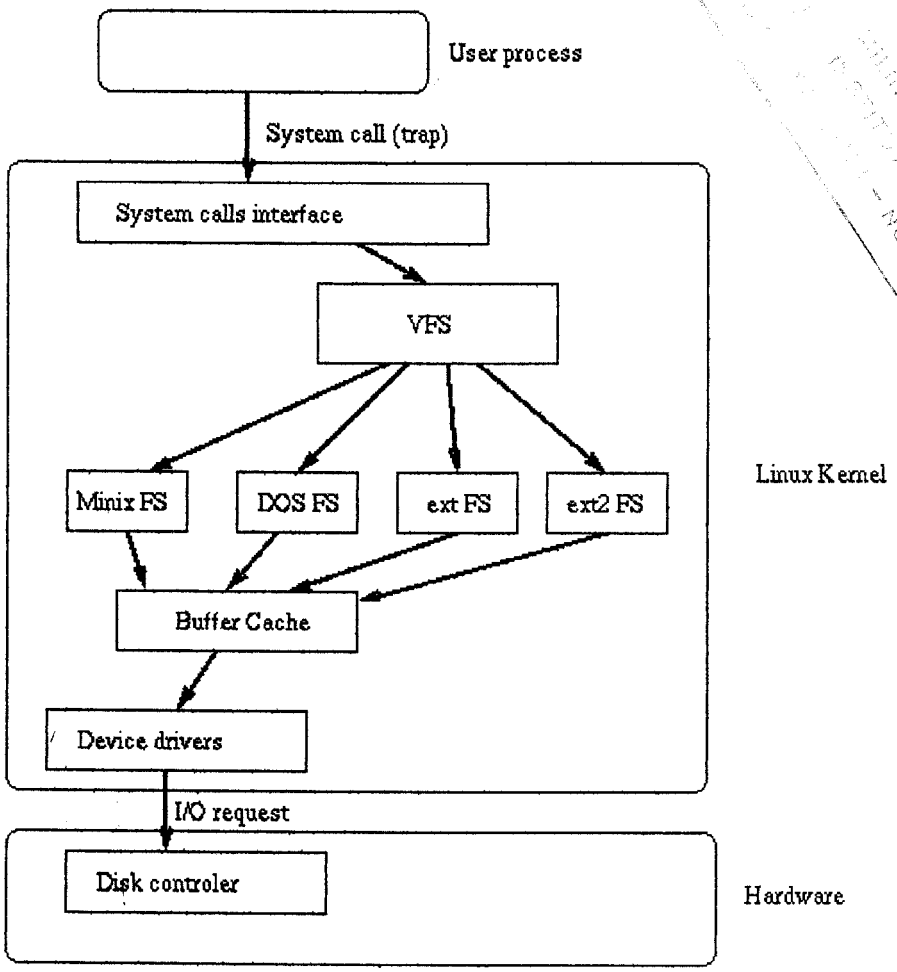
Gambar 2.6
Contoh struktur direktori yang berisikan tiga buah file

2.4 VIRTUAL FILE SYSTEM

Linux kernel berisi lapisan Virtual File System yang digunakan pada waktu system call berjalan pada file. VFS merupakan layer yang

menangani system call yang berorientasi pada file dan memanggil fungsi fungsi yang diperlukan pada file system fisik untuk melakukan proses I/O

Mekanisme ini digunakan untuk memudahkan integrasi dan penggunaan file system yang berbeda.



Gambar 2.7
Hubungan antara VFS dan file system yang lain

Ketika proses menjalankan system call file, kernel akan memanggil fungsi yang ada dalam VFS. Fungsi ini menangani struktur yang berdiri sendiri dan mengarahkan call pada fungsi yang terdapat

pada file system yang sebenarnya, yang bertanggung jawab untuk menangani file system itu sendiri.

Linux VFS digunakan supaya akses pada file menjadi lebih cepat dan efisien. VFS juga meyakinkan bahwa file dan data disimpan secara benar. Linux VFS meng-cache informasi dalam memori dari tiap file system pada waktu di mount dan sedang digunakan.

Gambar diatas menunjukkan hubungan antara VFS linux dengan file system yang sebenarnya. Virtual file system harus mengatur keseluruhan dari file system yang berbeda yang dimount kedalam system.

Ketika file system di inisialisasi, maka file system tersebut mendaftarkan dirinya pada VFS, hal ini terjadi ketika sistem operasi menginisialisasi pada waktu boot. File system yang sebenarnya bisa dibangun dalam kernel ataupun dibangun sebagai loadable module. Module file system di muat jika diperlukan saja. VFS menyimpan daftar dari file system yang dimount bersama dengan superblock VFS. Tiap superblock VFS berisikan informasi dan penunjuk yang menuju ke rutin yang melakukan fungsi-fungsi istimewa. Jika system memroses direktori dan file, rutin system akan dipanggil melintasi inode VFS dalam system. Virtual file system akan mencari diseluruh inode VFS yang merepresentasikan file system. Dikarenakan banyaknya proses seperti ini yang berulang ulang, maka inode inode tersebut akan disimpan didalam inode cache yang akan membuat akses menjadi lebih cepat.

Semua Linux file system menggunakan common buffer cache untuk melakukan cache buffer data dari device yang mendasari untuk

membantu mempercepat akses kearah device fisik yang ditempati file system. Buffer cache tidak tergantung pada file system dan terintegrasi pada mekanisme bahwa Linux kernel digunakan untuk mengalokasikan buffer data untuk proses read dan write.

Blok akan disimpan dalam global buffer cache waktu blok dibaca oleh file system. Buffer cache ini dipakai secara bersama oleh file system dan kernel linux. Buffer tersebut dikenali dari nomor bloknya dan pengenal unik untuk device yang membacanya. Jadi, jika data tersebut sering diakses maka data tersebut akan diambil dari buffer cache daripada dibaca dari disk.

VFS juga menyimpan cache direktori sehingga inode yang sering digunakan akan bisa cepat ditemukan.

2.4.1 SUPERBLOK VFS

Setiap file system yang telah dimount direpresentasikan oleh superblok VFS, diantara berbagai fungsi superblok VFS berisikan :

Device

Merupakan pengenal device untuk device blok dimana file system terdapat. Sebagai contoh, /dev/hda1 disk IDE pertama dalam disk mempunyai pengenal device 0x301

Inode Pointer

Inode pointer yang sudah dimount menunjuk pada inode pertama dalam file system. Inode pointer yang meliputi menunjuk pada inode yang merepresentasikan direktori dimana tempat file system.

Block Size

Ukuran blok dalam satuan byte untuk file system

Superblock Operation

Sebuah pointer yang menuju ke rutin superblok untuk file system. Diantaranya rutin rutin ini digunakan oleh VFS untuk membaca dan menulis inode dan superblok

Filesystem Type

Sebuah pointer yang menunjuk ke arah struktur data file system yang sudah dimount.

Filesystem Specific Information

Informasi yang dibutuhkan file system

2.4.2 INODE VIRTUAL FILE SYSTEM

Informasi dalam tiap inode VFS dibangun dari informasi dalam file system yang sebenarnya. Inode VFS terdapat hanya dalam memori kernel dan disimpan dalam inode cache VFS sepanjang berguna bagi system. Selain itu juga VFS berisikan field field berikut ini.

Device

merupakan pengenal device dimana file diletakkan.

Nomor inode

merupakan nomor inode dan bersifat unik dalam file system. Kombinasi dari device dan nomor inode unik dalam VFS.

Mode

field ini berisikan informasi tentang hak akses

ID user

pengenal pemilik (owner)

Waktu (time)

Waktu pembuatan, modifikasi, dan penulisan

Ukuran blok

Ukuran blok dari file dalam satuan byte.

Operasi inode

Sebuah pointer yang menuju pada alamat rutin. Rutin ini berbeda pada tiap file system dan melakukan operasi untuk inode.

Count

Banyaknya komponen system yang sedang menggunakan inode VFS. Jumlah 0 menunjukkan bahwa inode sedang tidak digunakan.

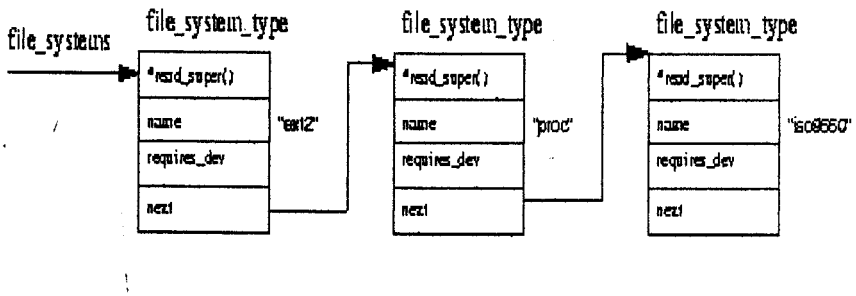
Lock

Field ini digunakan untuk mengunci (locking) inode VFS

Dirty

Apakah field telah berubah dari perubahan terakhir

2.4.3 MENDAFTARKAN FILE SYSTEM



gambar 2.8
File system yang telah terdaftar pada system

Kernel linux berisikan informasi tentang file system apa saja yang akan dimuat (diload) ketika system booting. File system linux bisa dibangun sebagai module dan bisa diminta untuk dimuat sepanjang dibutuhkan. Kapanpun file system module dimuat maka file system

tersebut akan terdaftar kedalam kernel, dan akan keluar jika file system di keluarkan dari kernel.

Pada waktu proses registrasi file system kedalam VFS dan direpresentasikan oleh struktur data file_system_type yang berisikan nama file system dan pointer yang menuju ke superblok VFS. Gambar 9.5 menunjukkan bahwa struktur file_system_type diletakkan dalam daftar yang ditunjuk oleh pointer file_system.

Rutin pembaca Superblock

Rutin ini dipanggil oleh VFS ketika sebuah instans file system di mount.

File System name

Nama dari file system, sebagai contoh adalah ext2

Device needed

Device yang dibutuhkan oleh file system

2.4.4 PROSES MOUNTING FILE SYSTEM

Ketika superuser mencoba untuk melakukan proses mount sebuah file system, kernel Linux harus pertama kali memvalidasi argumen argumen yang dilewatkan pada system call. Meskipun proses ini melakukan sejumlah cek dasar, akan tetapi proses ini tidak mengetahui file system mana dalam kernel yang telah dibangun untuk digunakan.

Misalkan pada perintah dibawah ini:

```
$ mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom
```

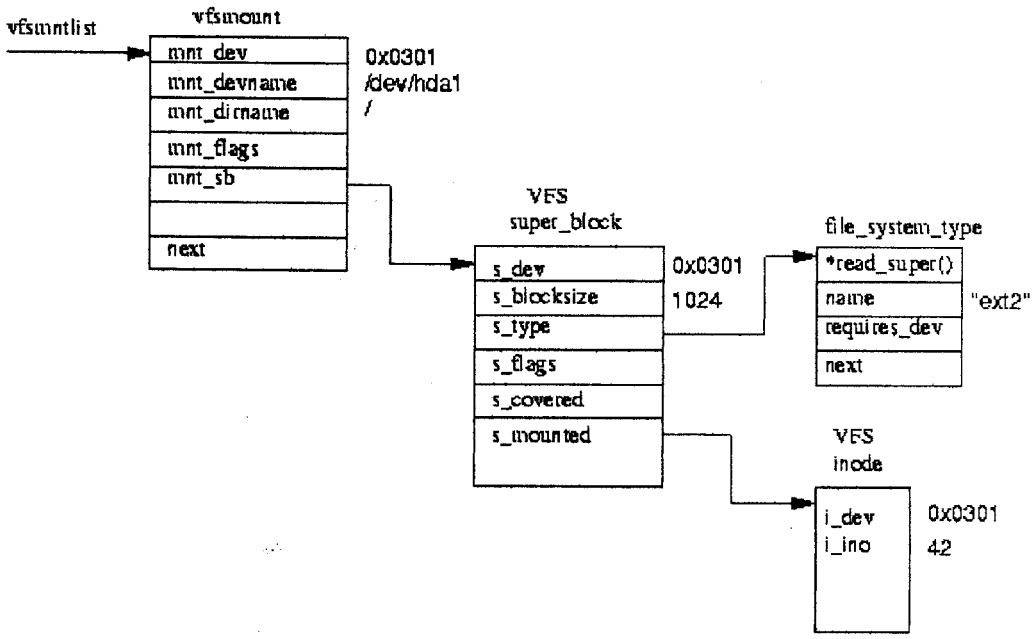
Perintah mount diatas akan melewatkan pada kernel tiga buah informasi yaitu nama file system, blok device yang berisi file system, dan ketiga adalah letak dimana file system tadi dimount di dalam topologi file system yang sudah ada.

Hal pertama yang harus dilakukan oleh Virtual File System adalah menemukan file system. VFS mecari diseluruh daftar file yang dikenali oleh file system dengan cara melihat struktur data file_system_type dalam daftar yang ditunjuk oleh file_system

Jika akhirnya ketemu maka akhirnya diketahui bahwa tipe file system telah didukung oleh kernel dan mempunyai alamat dari filesystem untuk membaca superblok file system.

Direktori yang sama tidak bisa digunakan sebagai mount point untuk lebih dari satu file system.

Rutin baca superblok harus mengisi superblok VFS berdasarkan dari informasi yang dibaca dari device fisik.



Gambar 2.9
Struktur file system yang telah dimount

Tiap tiap file system yang telah dimount dideskripsikan pada struktur data vfsmount. Yang mana dijadikan antrian dalam sebuah list yang ditunjuk oleh vfsmntlist

Pointer yang lain, vfsmntail menunjuk pada entri terakhir pada list dan mru_vfsmnt menunjuk pada file system yang sering digunakan

2.4.5 MENEMUKAN FILE DALAM VIRTUAL FILE SYSTEM

Untuk menemukan inode VFS sebuah file dalam VFS, VFS harus me-resolve nama direktori pada waktu itu, mencari inode VFS yang merepresentasikan tiap tiap direktori dalam nama. Tiap pencarian direktori melibatkan pencarian khusus dari file system yang mana alamatnya ditampung dalam inode VFS yang merepresentasikan direktori parent. Hal ini bekerja karena inode VFS dari dari root tiap file system

tersedia dan ditunjuk oleh superblok VFS untuk system tersebut. Tiap kali inode dicari oleh file system, maka ia akan mengecek pada direktori cache untuk direktori. Jika tiada entrinya maka file system tersebut akan mengambil VFS node dari file system sesungguhnya.

2.4.6 PROSES UNMOUNTING FILE SYSTEM

Sebuah file system tidak bisa diunmount jika sesuatu dalam file system sedang menggunakan salah satu file di dalamnya. Jika ada yang menggunakan file system yang akan dimount, maka akan ada inode VFS dalam VFS inode cache. Kode akan mengecek dengan cara mencari dalam daftar inode untuk mencari inode yang dimiliki oleh device tersebut. Jika hasilnya ternyata dirty maka ia akan dituliskan kembali pada disk. Lalu ia akan membebaskan entri tersebut dari list vfsmntlist.

2.4.7 INODE CACHE VFS

Ketika file system yang telah dimount dijelajahi, VFS inodenya akan secara terus menerus dibaca, dan dalam kasus tertentu ditulis. VFS menjaga inode cache untuk mempercepat akses pada keseluruhan file system yang sudah di mount.

Inode VFS cache diimplementasikan sebagai hash table yang mana entrinya adalah pointer pada daftar dari inode VFS yang mempunyai hash value yang sama. Hash value dari inode dihitung dari nomor inode dan dari pengenal device yang berisi file system. Kapanpun VFS membutuhkan akses sebuah inode yang pertamakali dilihat adalah dalam inode cache. Untuk menemukan sebuah inode dalam cache,

system harus pertama kali menghitung hash valuenya dan kemudian menggunakannya sebagai index pada inode hash table. Ini akan memberikannya sebuah pointer yang menunjuk pada daftar inode dengan hash value yang sama, yang kemudian akan membaca tiap inode sampai menemukan salah satu yang inode numbernya dan pengenal device-nya sama seperti yang dicari.

Jika inode bisa ditemukan dalam cache, maka parameter count bertambah untuk menunjukkan bahwa ada user lain yang memakai, dan akses file system berlanjut. Selain itu sebuah inode VFS yang free harus ditemukan sehingga file system bisa membaca inode dari memory. VFS mempunyai sejumlah pilihan tentang bagaimana mendapatkan inode yang free. Jika system telah mengalokasikan banyak inode VFS maka system akan mengalokasikan kernel page dan memecahnya kedalam inode yang baru , bebas dan meletakannya dalam daftar inode. Kesemuanya ada dalam sebuah daftar yang ditunjuk oleh first_inode sebagaimana halnya inode hash table.

Jika ada banyak inode, maka akan dipilih inode yang merupakan kandidat yang bagus, yang mana ditentukan dari jumlah count-nya yang 0, yang mengindikasikan bahwa tiada user yang sedang memakainya

Setelah inode VFS yang baru ditemukan, rutin spesifik file system harus dipanggil untuk mendapat indormasi yang dibaca dari file system. Ketika diakses maka VFS inode baru tersebut akan mempunyai count 1 dan dikunci sehingga tidak ada yang bisa mengakses sampai inode tersebut berisi informasi yang valid.

2.4.8 CACHE DIREKTORI

Untuk mempercepat akses pada direktori yang sering digunakan. VFS menggunakan cache dari entri direktori. Ketika direktori dicari oleh file system yang sebenarnya. Detailnya ditambahkan pada direktori cache, selanjutnya jika direktori yang sama dicari maka akan ditemukan dalam direktori cache. Hanya entri direktori yang pendek yang disimpan dalam cache (sampai 15 char).

Direktori cache terdiri dari hash table, tiap tiapnya menunjuk pada daftar entri direktori cache yang punya hash value yang sama. Fungsi hash menggunakan nomor device yang menampung file system dan nama direktori untuk menghitung offset, atau index ke dalam hash table. Ini menyebabkan direktori yang telah dicache bisa lebih segera ditemukan.

Untuk membuat supaya cache tetap valid dan terbaru maka VFS menyimpan daftar dari entri direktori cache yang paling sering digunakan terakhir (*Least Recently Used*). Ketika pertama kali entri direktori diletakkan dalam cache, yang kemudian akan dicari, maka akan ditambahkan kedalam bagian akhir dari level pertama LRU. Dalam full cache ini akan mengganti entri yang telah ada dari daftar LRU yang paling depan. Ketika entri direktori diakses lagi, maka ia akan dipromosikan di belakangnya. Sehingga dapat disimpulkan bahwa entri yang terletak di depan adalah entri yang jarang diakses.

2.5 MODUL FILESYSTEM

Linux merupakan sistem operasi yang menggunakan kernel monolithic, yang merupakan sebuah program yang besar yang kesemua komponennya mempunyai akses ke seluruh struktur data dan rutin internal. Ada model lain yang merupakan struktur mikro-kernel dimana bagian yang fungsional dipecah pecah kedalam unit yang terpisah dengan komunikasi yang dibatasi antara mereka.

Hal ini menjadikan setiap kali penambahan komponen baru kedalam kernel melalui proses konfigurasi agak merepotkan, karena setiap kali ada komponen baru maka kernel harus dibuild dahulu baru bisa digunakan

Linux memperbolehkan memuat ataupun melepaskan komponen dari sistem operasi sepanjang dibutuhkan. Module linux merupakan sebuah kode yang bisa dilink kedalam kernel kapan saja dibutuhkan dan tidak membutuhkan restart. Module bisa dilepaskan dari kernel jika tidak lagi dibutuhkan. Kebanyakan module kernel linux merupakan device driver, atau file system.

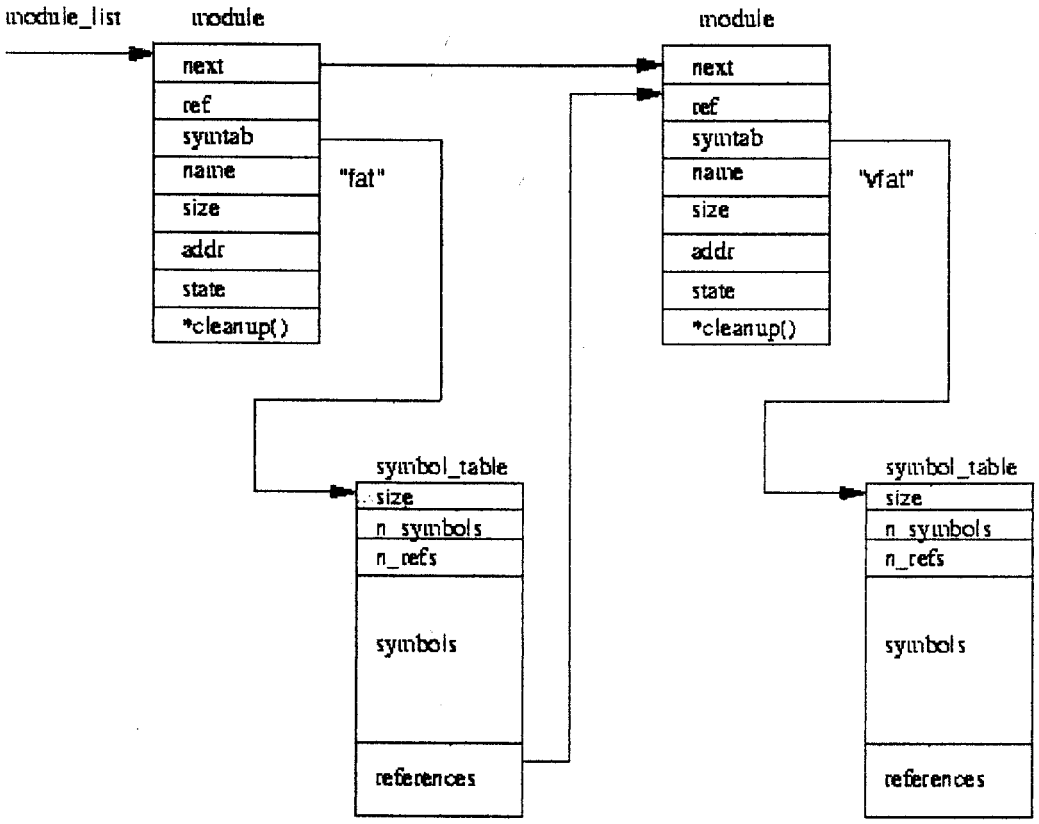
Linux kernel bisa dimuat atau dilepaskan dengan menggunakan perintah insmod atau rmmod, atau menggunakan kernel daemon.Kedinamisan memuat kode ketika diperlukan saja membuat ukuran kernel menjadi kecil dan menjadikannya lebih fleksibel. Module juga bisa berguna untuk mencoba kode kernel baru tanpa harus membangun ulang dan me-restart kernel setiap kali ingin mencoba.module membutuhkan struktur data tambahan yang membutuhkan sedikit memory.

Jika module linux sudah dimuat, maka module tersebut telah menjadi bagian kernel seperti halnya kernel normal. Kode module mempunyai hak dan tanggung jawab yang sama seperti halnya kode kernel yang ada, sehingga module tersebut bisa saja mengakibatkan crash sistem operasi. Kernel menyimpan daftar resource kernel dalam tabel simbol kernel sehingga bisa menemukan kembali referensi pada resource tersebut dari module pada waktu dimuat. Linux mengijinkan tumpukan module, yang mengijinkan module untuk membutuhkan module yang lain.

Satu module yang membutuhkan service atau resource dari module yang lain sama dengan keadaan dimana jika sebuah module membutuhkan service atau resource dari kernel. Ketika tiap module dimuat, kernel akan memodifikasi kernel symbol table, dan menambahkan resource atau simbol yang dikirimkan oleh module yang baru dimuat. Hal ini berarti bahwa ketika module berikutnya dimuat, maka module tersebut akan mempunyai akses ke module yang dimuat sebelumnya.

Ketika mencoba untuk melepaskan module dari kernel, maka kernel perlu untuk mengetahui bahwa module telah tidak terpakai dan membutuhkan cara untuk memberikan tanda bahwa module akan dilepaskan. Sehingga module akan bisa untuk membebaskan resource system yang telah dialokasikan.

2.5.1 PEMBUATAN MODUL



Gambar 2.10
Struktur modul kernel dalam memory

Ada dua jalan untuk memuat modul kedalam kernel, cara pertama adalah menggunakan perintah insmod yang merupakan cara manual untuk memasukkan sebuah modul kedalam kernel. Cara kedua adalah memasukkan modul hanya ketika dibutuhkan. Yang dikenal dengan cara demand loading. Ketika sebuah kernel membutuhkan module, maka

kernel akan meminta kernel daemon untuk mencoba memuat module yang dibutuhkan.

Kernel daemon adalah sebuah proses yang mempunyai hak akses super user. Ketika mulai dijalankan, biasanya pada waktu boot. Kerneld akan membuka IPC (Inter Process Communication) pada kernel. Link ini digunakan oleh kernel untuk mengirimkan pesan kepada kerneld untuk meminta task yang akan dikerjakan. Fungsi utama Kerneld adalah untuk memuat dan melepaskan modul kernel dan juga berkemampuan yang lain seperti dial on demand.

Kerneld tidak melakukan task ini sendiri, akan tetapi kerneld menjalankan program yang diperlukan seperti insmod untuk melakukannya. Kerneld hanyalah sebuah agent dari kernel.

Program utility insmod harus menemukan modul kernel yang akan dimuat. Module yang diload biasanya berada di direktori /lib/modules/kernel-version

Modul kernel ini merupakan object file yang dilink seperti hanya program lain dalam system. Yang membedakannya adalah bahwa program module tersebut adalah sebagai image yang bisa dipindah pindah letaknya.

Program program tersebut disimpan dalam pasangan pasangan yang berisikan nama simbol dan nilainya. Tabel simbol kernel yang sudah dieksport disimpan dalam struktur data modul yang pertama dalam daftar modul yang dibuat oleh kernel dan ditunjuk oleh pointer module_list.

Hanya simbol yang dimasukkan secara khusus yang dimasukkan kedalam tabel yang mana dibangun ketika kernel dicompile dan dilink,

tidak setiap symbol dalam kernel dieksport pada module. Sebagai contoh symbol adalah "request_irq" yang mana rutin kernelnya harus dipanggil ketika driver ingin mengambil kontrol dari interupt system dasar.

Symbol kernel yang telah diexport dan nilainya bisa dilihat dengan cara melihat file dalam /proc/ksysms atau menggunakan program ksyms. Ksyms bisa memperlihatkan keseluruhan dari simbol kernel ataupun simbol yang dieksport oleh modul. Insmod akan membaca module ke dalam virtual memory. Penetapan ini membentuk module image dalam memory. Insmod secara fisik akan menuliskan alamat simbol kedalam tempat yang dibutuhkan dalam modul.

Ketika insmod menetapkan referensi modul kedalam simbol kernel, maka insmod akan meminta tempat dalam kernel untuk menampung kernel baru menggunakan system call. Kernel mengalokasikan struktur data modul baru dan memori kernel untuk menampung modul baru dan meletakkannya pada akhir dari daftar modul kernel. Modul yang baru kemudian akan ditandai dengan tanda UNINITIALIZED

Gambar 2.10 memperlihatkan daftar dari modul kernel setelah dua modul, yaitu VFAT, dan VFAT telah dimuat kedalam kernel.

Modul baru juga mengeksport simbol ke kernel dan insmod akan membangun sebuah tabel dari simbol ini. Setiap modul kernel haruslah berisi rutin inisialisasi dan rutin untuk mengeluarkan modul dari kernel.

Ketika modul baru ditambahkan dalam kernel, modul tersebut harus mengupdate kumpulan simbol dan merubah modul yang digunakan oleh modul baru. Modul yang mempunyai modul lain yang bergantung

padanya harus melihat daftar referensi di akhir tabel simbol mereka dan ditunjuk oleh modul data strukturnya

Gambar 2.6 memperlihatkan bahwa file system VFAT bergantung pada file system FAT. Sehingga module FAT berisikan referensi pada module VFAT; referensi ditambahkan ketika module VFAT dimuat. Kernel memanggil rutin inisialisasi modul. Alamat rutin cleanup diletakkan di struktur data modul dan akan dipanggil oleh kernel ketika modul diunload, setelah ini maka keadaan modul akan diset menjadi RUNNING

2.5.2 PENGHAPUSAN MODUL

Module bisa dikeluarkan menggunakan command rmmod secara manual. Atau jika menggunakan kerneld, maka modul akan secara otomatis dikeluarkan dari kernel jika sudah tak lagi digunakan. Setiap kali pewaktu melampai batas yang sudah ditentukan, maka kerneld akan menjalankan system call yang meminta agar semua modul yang telah dimuat tersebut dikeluarkan dari system.

Module tidak bisa diunload sepanjang ada module lain dalam kernel yang menggunakannya. Sebagai contoh adalah VFAT module tak akan bisa dikeluarkan dari kernel jika ada file system VFAT yang sedang dimount. Penggunaan perintah lsmod akan menampilkan apakah modul tersebut sedang dipakai atau tidak.

```
Module:                 #pages:                 Used by:
msdos                   5                       1
vfat                    4                       1 (autoclean)
fat                     6 [vfat msdos]          2 (autoclean)
```

Count merupakan jumlah entitas kernel yang bergantung pada modul ini. Di contoh diatas, vfat dan modul msdos keduanya bergantung pada modul fat dan mempunyai count 2. Keduanya vfat dan modul msdos mempunyai satu modul yang memakainya, yang merupakan file system.

Struktur data modul ditandai dengan tanda DELETED dan diunlink dari list modul kernel. Setiap modul lain yang bergantung pada daftar referensi diubah sehingga

BAB III

PARALEL VIRTUAL FILE SYSTEM

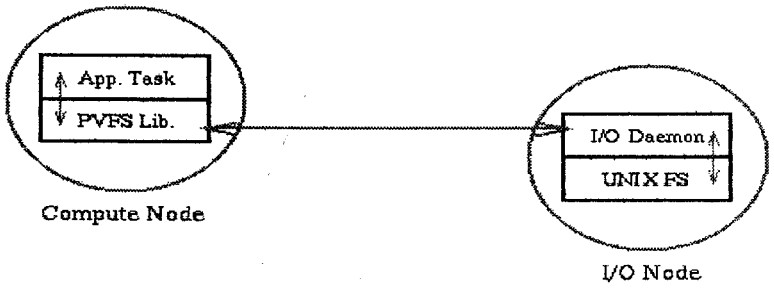
Sebagai sebuah paralel file system, maka PVFS haruslah menyediakan akses berkecepatan tinggi pada data, sebagai tambahan PVFS menyediakan name space yang konsisten di semua kluster yang merupakan bagian dari PVFS, dan juga memungkinkan pengguna untuk menentukan striping data di berbagai I/O node dan memungkinkan file file binary yang sudah ada untuk dijalankan seperti halnya dijalankan di file system yang ada.

3.1 ARSITEKTUR PARALEL VIRTUAL FILE SYSTEM

PVFS merupakan kumpulan dari daemon dan modul yang memungkinkan aplikasi untuk mengakses file yang diletakkan di banyak disk / mesin secara paralel. Sistem operasi yang digunakan baru terbatas pada Linux.

PVFS didesain sebagai sistem client server dengan banyak server, yang disebut dengan I/O daemon. I/O daemon berjalan di node yang terpisah dalam cluster, yang disebut I/O nodes, yang mempunyai disk di masing masing mesin. Tiap file PVFS di strip di berbagai disk dalam I/O node.

Gambar 3.1 berikut ini menggambarkan skema pengiriman data antara client dan server

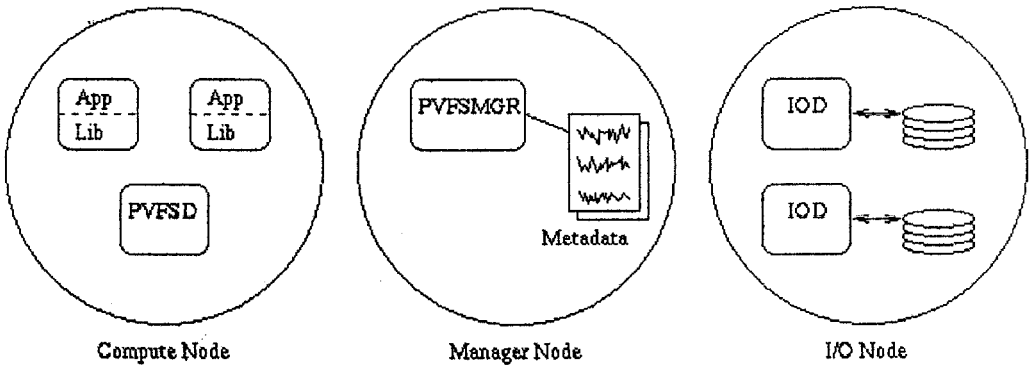


Gambar 3.1
Sistem pengiriman data pada PVFS

PVFS dibagi menjadi tiga jenis, yang pertama merupakan client (pengguna), I/O node dan node untuk manajemen. Node tunggal bisa menjadi satu atau lebih dari jenis ini. Bisa saja node tunggal menjadi ketiga tiganya sekaligus.

PVFS mempunyai beberapa daemon dan modul untuk mengakses file system. Ada dua jenis daemon yanitu management daemon, dan I/O daemon. Jenis yang normal adalah ada sebuah daemon manajemen (yang berjalan di node manajemen) dan beberapa I/O daemon (yang berjalan dimana I/O berada). Modul yang digunakan aplikasi berjalan di client untuk berkomunikasi dengan management daemon dan I/O daemon.

Managemen daemon mempunyai dua tanggung jawab, yaitu memvalidasi hak akses untuk mengakses file dan membuat metadata pada file file PVFS. Hanya diperlukan satu management daemon untuk melakukan operasi file system dan sebuah management daemon bisa mengatur berbagai file system. Sedangkan I/O daemon melayani akses pada file data PVFS di masing masing node yang diminta oleh client.



Gambar 3.2
Jenis Jenis Node Pada PVFS

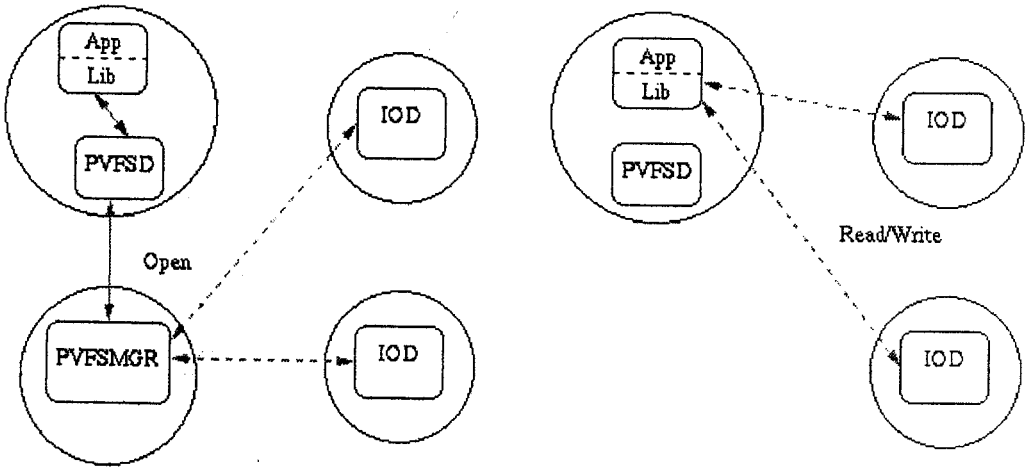
PVFS memungkinkan direktori yang akan dimanage sebagai file system yang terpisah. Tiap file system harus mempunyai satu node I/O node harus mempunyai direktori dalam disk lokal untuk menyimpan file data. I/O node bisa menyimpan file data ini pada file system yang telah ada.

Management node (manager) inilah yang nantinya akan digunakan oleh workstation-workstation untuk menggunakan file system tersebut.

Ketika sebuah file dibuka oleh aplikasi, sebuah request dilewatkan pada PVFSD dalam mesin. PVFSD menentukan file system termount apa sajakah yang akan diakses dan melewatkan request yang ada pada PVFSMGR untuk file system tersebut. Manager kemudian menentukan IOD yang mana saja yang mempunyai data untuk file tersebut (dengan cara melihat pada file metadata). Akhirnya file dibuka oleh semua IOD. Alamat dari IOD kemudian dilewatkan kembali pada aplikasi. Sekali file terbuka, semua akses pada file ini akan mengambil tempat dengan cara berkoneksi langsung pada IOD itu sendiri. Koneksi terjadi selama akses

berlangsung dan PVFSD dan PVFSMGR tidak terlibat lagi sampai file tertutup.

Rutin rutin PVFS membuat struktur data untuk menangani pembagian pada tiap file, dan mengatur rantai komunikasi antara aplikasi dan IOD, dan melakukan pengumpulan dan pengaturan data ke atau dari IOD. Berikut merupakan cara kerja PVFS



Gambar 3.3
Diagram Cara Kerja PVFS

3.2 PVFS MANAGER DAN METADATA

Manager membuat, menghapus dan mengubah file metadata yang diletakkan di dalam direktori metadata. File file ini berukuran kecil dan hanya berisikan informasi tentang hak akses file, lokasi data dan bukan data itu sendiri.

I/O daemon menuliskan file data pada direktori dalam node lokal di file yang bernama "fXXXXXX.Y", dimana X merepresentasikan nomor inode dari file (inode file metadata) dan Y adalah index file yang sedang digunakan. File file ini disimpan dengan hak akses yang diset sehingga bisa dibaca oleh I/O daemon.

Direktori ini dibuat dan diatur secara otomatis oleh I/O daemon dan akan berisikan keseluruhan file data. Modul PVFS akan berinteraksi dengan manager dan I/O daemon untuk melakukan proses proses I/O. I/O daemon akan menggunakan file metadata untuk menentukan bagaimana menghubungi manager yang bersangkutan ketika mengakses file. Jika manager telah memberikan hak akses untuk mengakses file, maka koneksi akan terjadi dengan I/O daemon yang melakukan operasi I/O pada aplikasi. Keseluruhan hal ini akan ditangani oleh modul file system PVFS. Aplikasi akan mengakses file system PVFS seperti halnya mengakses file system yang lain seperti MSDOS, ext2fs Management daemon bertanggung jawab pada penyimpanan dan akses pada seluruh metadata dalam PVFS. Metadata, dalam konteks file system, mengacu pada informasi yang menjelaskan karakteristik file

seperti permission, owner dan groupnya, dan yang lebih penting lagi adalah letak file data dalam distribusinya. Dalam kasus paralel file system, informasi tersebut harus memasukkan unsur lokasi file dalam disk dan lokasi disk dalam kluster. Dalam PVFS, file data dan metadata disimpan dalam file di file system yang sudah ada.

File file PVFS di strip diseluruh kumpulan I/O node untuk keperluan akses paralel. Distribusi file ditunjukkan dengan tiga parameter metadata yaitu nomor I/O node mula mula, jumlah I/O node dan ukuran strip. Parameter ini bersama dengan urutan penempatan I/O node untuk file system, akan bisa menyatukan file file yang terdistribusi tersebut menjadi lengkap. Sebagai contoh adalah untuk metadata file bernama /pvfs/foo yang ada di tabel 3.1.

Field pcount menentukan bahwa data disebar dalam I/O node, base menentukan bahwa I/O node pertama adalah I/O node nomor 2 ,dan ssize menentukan bahwa ukuran stripnya adalah 64 Kbyte. User bisa merubah parameter ini ketika file dibuat.

Inode	**1092157504**
Base	**2**
Pcount	**3**
ssize	**65536**

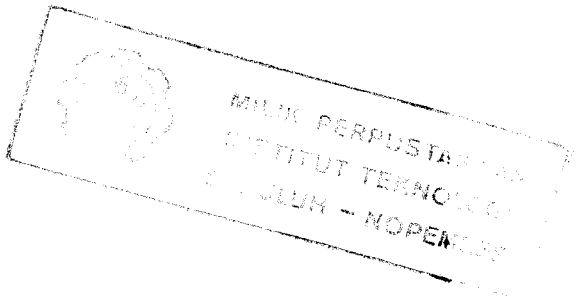
Tabel 3.1
Contoh Metadata: File /pvfs/foo

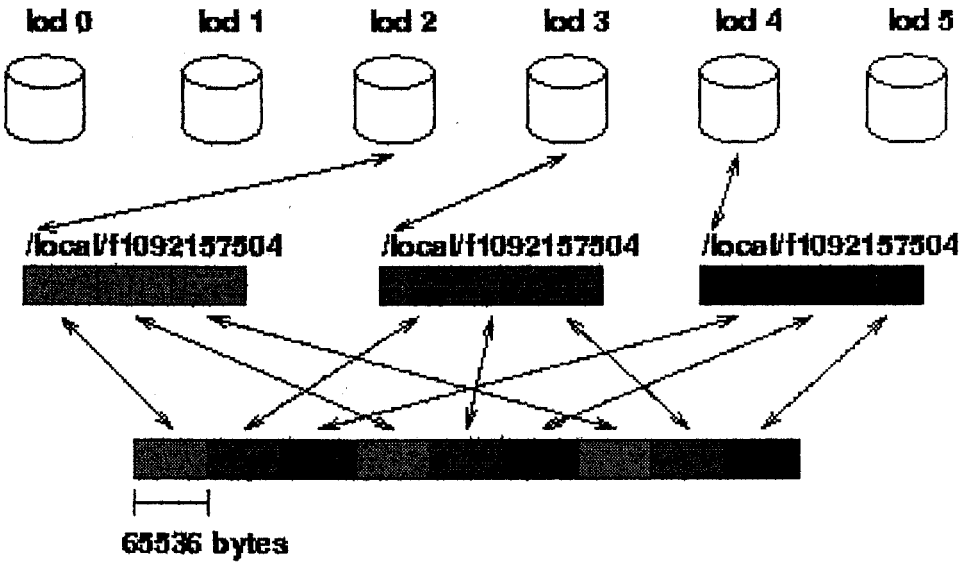
Aplikasi berkomunikasi dengan PVFS manager dengan TCP ketika menjalankan operasi file seperti membuka file, membuat file, menutup file dan menghapus file. Ketika aplikasi membuka file, maka manager akan mengembalikan pada aplikasi lokasi dari I/O node dimana file data diletakkan. Informasi ini mengijinkan aplikasi untuk berkomunikasi dengan I/O node ketika file data diakses.Lain kata manager tidak akan dihubungi pada waktu operasi baca/tulis.

3.3 I/O DAEMON DAN PENYIMPANAN DATA

Pada waktu file system diinstal, user menentukan node mana dalam kluster yang akan dijadikan server sebagai I/O node. Urutan I/O daemon PVFS berjalan pada I/O node. I/O daemon bertanggung jawab dalam mengakses localdisk dimasing masing I/O node untuk menyimpan file data.

Gambar berikut ini memperlihatkan file /pvfs/foo didistribusikan pada PVFS yang didasarkan pada tabel 1. Dalam contoh ada 6 buah I/O node, dan file filenya distrip hanya pada tiga I/O node, dimulai dari node 2, karena file metadata menentukan strip tersebut. Tiap I/O daemon menyimpan bagian PVFS file tersebut dalam file di local system dalam I/O node. Nama file ini berdasarkan pada nomor inode yang ditentukan oleh manager pada file PVFS.





Gambar 3.4
Sebuah strip file

Seperti yang telah disebutkan, ketika aplikasi (client) membuka file PVFS, PVFS manager akan memberitahukan lokasi I/O daemon. Client kemudian akan membuat koneksi langsung pada I/O daemon. Ketika client bermaksud untuk mengakses data, client akan mengirimkan file deskriptor dari file yang akan diakses pada I/O daemon yang menyimpan data tersebut. Daemon menentukan bagian mana dari file tersebut yang disimpan secara lokal dan kemudian melakukan I/O yang diperlukan dan kemudian melakukan transfer data.

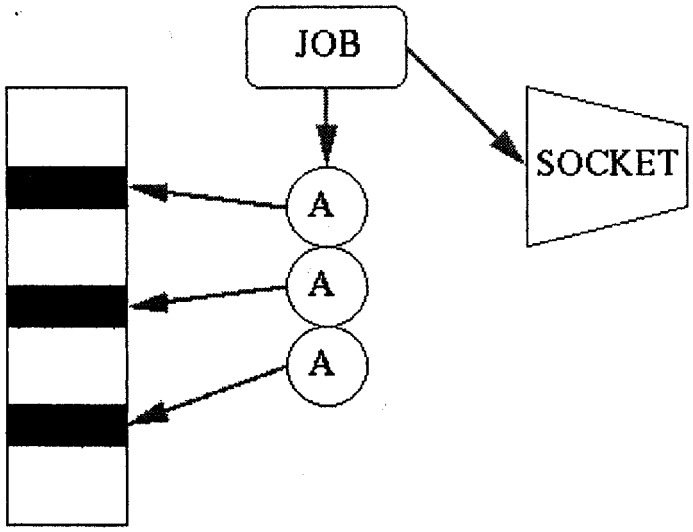
3.4 PEMROSESAN REQUEST

Ketika sebuah IOD menerima request, ia akan mengkonversikan parameter request kedalam sebuah struktur job yang terdiri dari job header dan struktur dari daftar akses. Tiap struktur akses

merepresentasikan sebuah segmen dari data yang tertransfer dari segmen terpotong yang menuju ke aplikasi melalui socket TCP/IP. Tiap group logik pada request diproses dalam urutan menurut parameter request. Untuk memroses tiap group, IOD akan menentukan lokasi file apakah secara logik file tersebut terletak pada IOD tersebut.

Jika tidak, maka IOD akan mencari lokasi logik file tersebut berikutnya yang terletak pada satu dari potongan letak file tersebut, dan jika merupakan batas dari group, maka proses akan dilakukan dari tempat tersebut. Kemudian IOD akan memeriksa untuk melihat apakah group juga terletak diluar dari potongan stripe file. Jika iya maka ia akan membangun sebuah struktur akses dari lokasi fisik sampai akhir potongan stripe, selain itu maka struktur akses akan dibangun sampai akhir group. Proses ini berulang sampai semua data dalam group yang berasosiasi dengan IOD telah direferensi oleh sebuah akses.

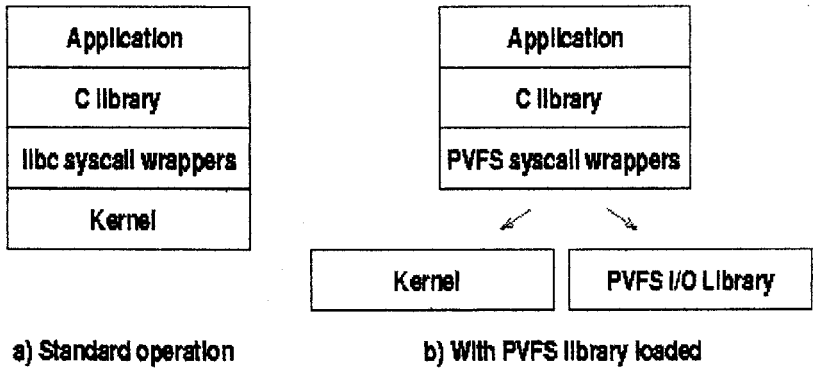
Struktur job digunakan oleh IOD untuk mengarahkan pembacaan disk block dan transfer data pada aplikasi. IOD akan melayani banyak request menggunakan struktur job sebagai sebuah catatan proses untuk tiap-tiap request. Data transfer bisa ditunjukkan menggunakan berbagai teknik termasuk memory mapped files, buffered I/O, raw I/O dan asynchronous I/O.



Gambar 3.5
Struktur job PVFS

3.5 I/O TRAPPING

System call merupakan fungsi low-level yang digunakan oleh aplikasi untuk berinteraksi dengan kernel (sebagai contoh, untuk disk, dan I/O). system call ini dibuat oleh fungsi yang diimplementasikan di library C standard, yang mana menangani passing parameter dalam kernel. Cara yang lebih maju adalah menangkap system call dengan cara menyediakan library yang terpisah. Sepeti memakai module



Gambar 3.6
Skema I/O trap

Ketika mengkompilasi aplikasi, salah satu cara praktis untuk menghindari ukuran file yang besar adalah dengan menggunakan dynamic linking. Sehingga file file tersebut akan bisa mendukung fungsi yang sama tanpa mengkompilasi ulang. Cara ini bisa digunakan pada PVFS client untuk menangkap I/O system call sebelum di lewatkan pada kernel. PVFS membuat sebuah library untuk mengakses yang dimuat kedalam memory sebelum library C yang standard diakses. Variabel Environment LD_PRELOAD dipakai untuk keperluan ini.

Gambar 3.6a memperlihatkan pengaturan dari mekanisme system call sebelum library dimuat ke memory ketika menjalankan aplikasi. Sedangkan gambar 3.6b memperlihatkan pengaturan dari mekanisme system call yang meletakkan library diantara kernel dan C library yang standard. Dalam kasus ini library tersebut diganti oleh PVFS yang menentukan tipe file apa yang akan diakses , jika file tersebut adalah PVFS maka I/O library PVFS yang akan digunakan untuk menangani fungsi.

Metode ini mempunyai kelemahan, yaitu jika library standar dari system berubah, maka kode library PVFS juga harus berubah, sehingga

metode ini tidak cukup fleksibel untuk pemindahan di lain mesin / lain sistem operasi.

3.6 MODUL VFS

Module merupakan bagian dari kernel yang bisa dimuat sepanjang dibutuhkan oleh user. Penggunaan modul bisa mengatasi kelemahan yang ditunjukkan oleh trapping pada library. Modul memungkinkan pengaksesan PVFS seperti pengaksesan file system yang lain di linux seperti CDROM, MSDOS dan sebagainya. Modul tersebut nantinya hanya berinteraksi dengan VFS yang merupakan sistem dasar file system di Linux. Setelah dimount, maka file PVFS tersebut akan bisa diakses seperti halnya mengakses file yang terdapat di file system yang lain.

Penggunaan modul selain sangat fleksibel juga memungkinkan untuk pengembangan yang tidak seperti halnya pada library yang menggunakan I/O trapping dimana library PVFS akan dimuat dulu sebelum proses I/O yang sebenarnya berjalan.

Kelemahan dari I/O trapping tersebut adalah jika library standar dari sistem operasi berubah, maka library PVFS juga harus diubah kodenya menyesuaikan dengan library standar tersebut. Kelebihan penggunaan modul VFS adalah, bahwa kode memggunakan metode VFS yang memang ditujukan untuk fleksibilitas file system. Yang mana setiap file system baru di Linux akan menggunakan VFS agar bisa digunakan dan dipakai oleh kernel Linux.

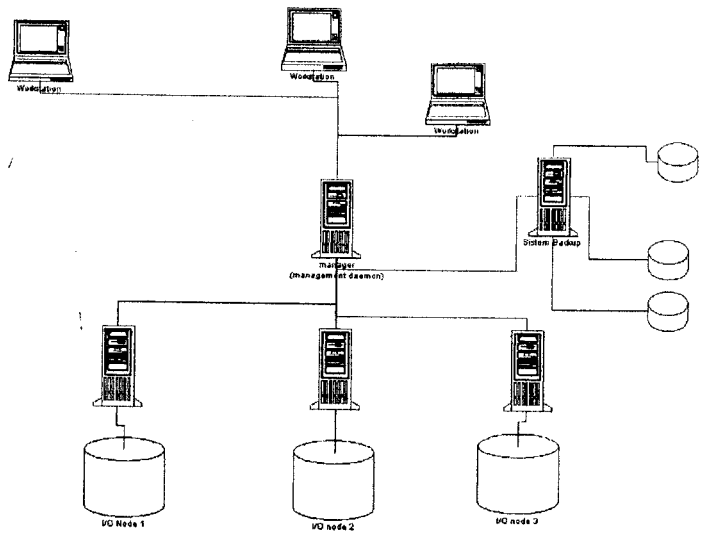
Walaupun demikian, kedua cara tersebut telah ada dan disediakan untuk mengakses file system PVFS tersebut.

BAB IV

PENGEMBANGAN SISTEM BACKUP PADA PARALEL VIRTUAL FILE SYSTEM

PVFS didesain dengan tidak mendukung sistem redundansi. Jika salah satu I/O node gagal beroperasi, maka sistem PVFS akan down dan tidak dapat digunakan untuk menyimpan data. Oleh karena itu maka diperlukan sebuah sistem backup untuk mengatasi hal tersebut.

4.1 ARSITEKTUR SISTEM BACKUP

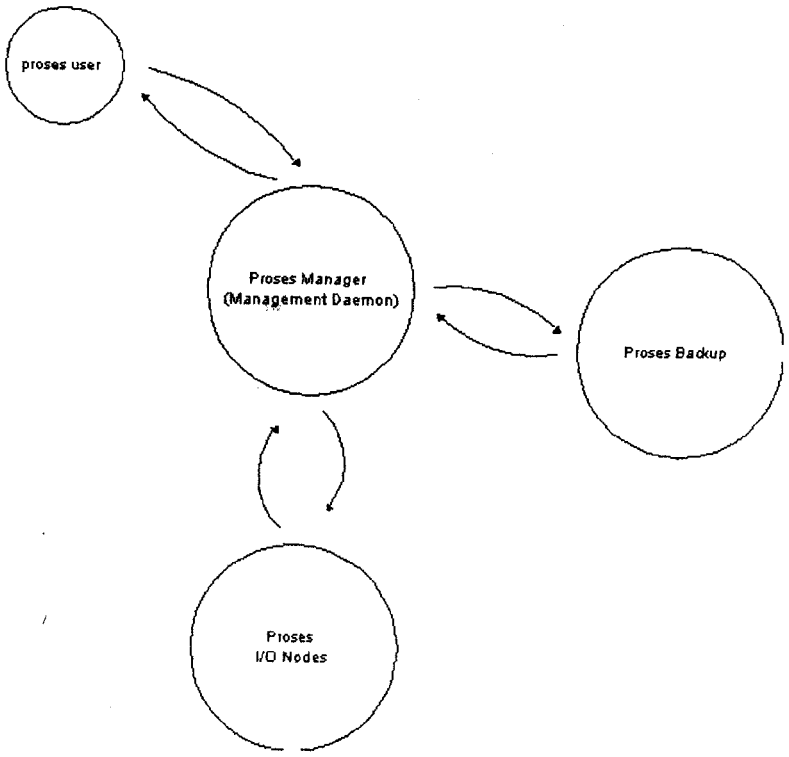


Gambar 4.1
Sistem backup pada PVFS

Pada perancangan sistem backup ini, ditambahkan sebuah mesin lagi yang akan berfungsi sebagai I/O daemon (IOD) sesuai dengan

jumlah I/O daemon yang ada yang akan ditempatkan pada mesin tersebut. Management node (manager) pada setiap operasinya pada sebuah i/o node akan melakukan operasi yang sama pada node backup. Sehingga ada dua proses yang sama yang dilakukan oleh manager yang pertama adalah menulis/membaca pada i/o node dan yang kedua adalah menulis/membaca pada backup node.

Data flow diagram dari proses backup adalah sebagai berikut

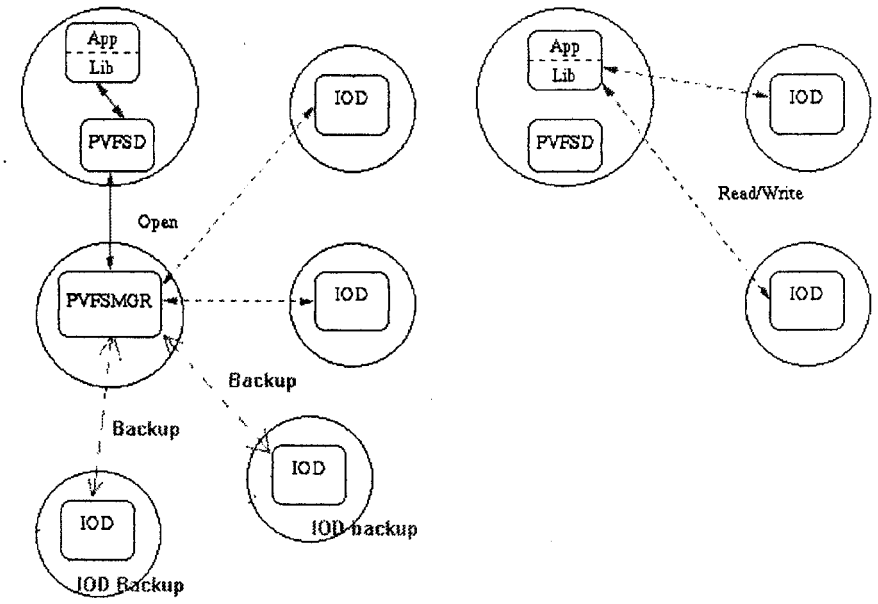


Gambar 4.2
DFD sistem backup

Backup node harus menjalankan I/O node (iod) sesuai dengan jumlah i/O node yang sama pada sistem PVFS. Untuk pengimplementasiannya maka pada host yang berfungsi sebagai manager ditambahkan sebuah konfigurasi baru yang akan mencatat host mana yang berfungsi sebagai sistem backup. Manager akan

mengidentifikasikan apakah I/O node yang bersangkutan down atau tidak jika tidak maka manager akan mengambil dari I/O yang bersangkutan dan jika tidak maka I/O node akan dicatat dan dialihkan pada sistem backup.

Jika dilihat dari bab III diatas, maka cara kerja sistem akan bertambah sebagai berikut :



Gambar 4.3
Cara kerja PVFS + sistem backup

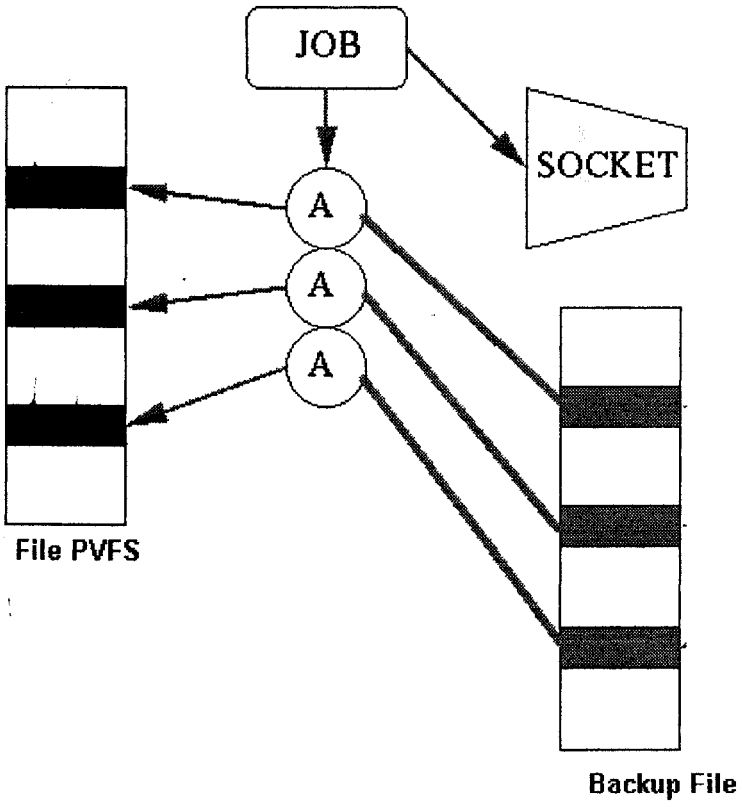
Perubahan yang terjadi adalah bahwa PVFSMGR akan melakukan proses sebanyak 2 x jumlah node yang terletak pada cluster PVFS. Jika IOD berjumlah 2 buah maka proses yang terjadi adalah empat kali. Karena ada dua buah IOD biasa dan dua buah IOD backup.

Kesesuaian jumlah IOD dan IOD backup merupakan suatu keharusan agar proses backup berjalan dengan baik.

4.2 PEMROSESAN SISTEM BACKUP

Seperti yang telah dijelaskan pada bab sebelumnya, bahwa pemrosesan pada PVFS menggunakan sebuah struktur job. Struktur job merupakan sebuah rantai struktur yang bertugas menampung dan mengatur antrian pada file system.

Pada sistem backup, modifikasi dilakukan pada pemrosesan jobnya. Proses job akan dilakukan dua kali yang pertama adalah penulisan ke node awal, dan yang kedua adalah penulisan ke node backup.

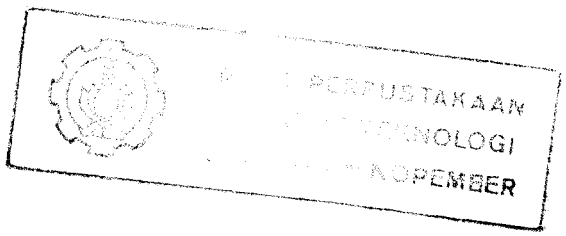


Gambar 4.4
Arsitektur Sistem backup

Proses ini dilakukan pada semua rutin yang menangani perubahan pada file yang terletak di IOD. Sedangkan pada pembacaan

hal ini tidak dilakukan. Sedangkan rutin rutin yang menangani proses pada manager tidak dimodifikasi karena hanya menangani perubahan file pada manager.

Pada intinya sistem backup hanya menduplikasi proses yang terjadi antara manager dan IOD. Akan tetapi sistem backup juga perlu dikonfigurasi. Sistem backup mempunyai file konfigurasi sendiri yang nantinya harus diset bersesuaian dengan IOD yang dipakai oleh manager.



BAB V

IMPLEMENTASI PVFS

Pada bab ini akan dibahas mengenai Implementasi dari Paralel Virtual File System, yang dijalankan pada sistem operasi linux.

5.1 RANCANGAN

Sistem Paralel virtual file system dibangun dengan menggunakan mesin mesin dengan sistem operasi linux, untuk saat ini. Paralel virtual file system membutuhkan mesin mesin dengan sistem operasi linux, dengan kernel 2.2.x dan glibc versi 2 keatas.

Rancangan virtual file system di jurusan teknik informatika diimplementasikan pada dua buah mesin. Mesin pertama berfungsi sebagai I/O daemon. Sedangkan mesin kedua berfungsi sebagai I/O daemon sekaligus manager.Pemakai bisa menggunakan pvfs dengan mesin linux yang lain yang telah diinstall modul file system pvfs.

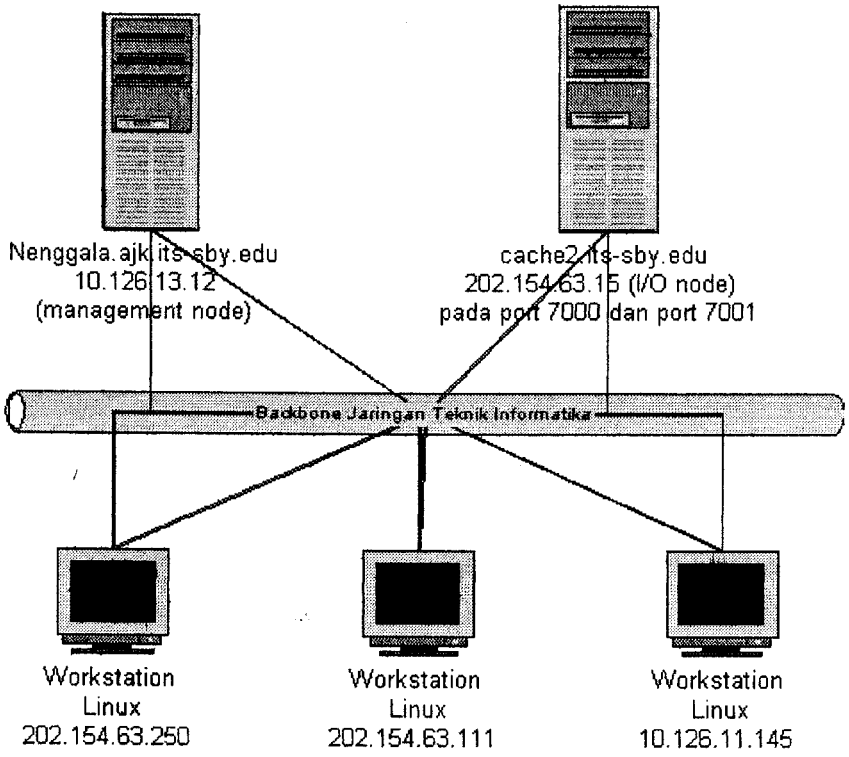
Manager tidak harus terpisah dari I/O node, karena masing masing berjalan sebagai servis yang mempunyai port tersendiri.

Pada rancangan ini ada tiga buah server yang saling berhubungan. Dua buah server berfungsi sebagai I/O daemon, dan satu server menjadi management node. Management node inilah yang nantinya menjadi tempat untuk melakukan proses penyusunan data dari I/O node I/O node yang ada dan juga bertanggung jawab pada hak akses file pada pvfs.

Server yang dimaksud pada paragraf diatas tidaklah harus merupakan satu mesin tersendiri. Server merupakan sebuah program yang di *bind* pada port tertentu dan me-listen request.

Pada rancangan ini dipakai tiga buah mesin untuk pengimplementasian paralel virtual file system. Dua buah mesin bertindak sebagai I/O nodes, dan ada satu mesin yang berfungsi sebagai manager sekaligus I/O node, dan sebuah mesin lagi untuk mesin backup.

Skema rancangan adalah sebagai berikut :



Gambar 5.1
Rancangan implementasi PVFS

5.2 RANCANGAN PERANGKAT LUNAK

Paralel Virtual File System dijalankan pada sistem dengan asumsi bahwa

1. Akses root pada file system lokal tersedia pada manager daemon untuk aktivitas managemen
2. Antar node mendukung jaringan dengan protokol TCP/IP
3. Ruang harddisk tersisa pada I/O node.

PVFS mengijinkan berbagai direktori untuk dianggap sebagai file system yang terpisah. Tiap file system harus mempunyai sebuah node yang ditugaskan sebagai node management dan ada juga yang bertugas sebagai I/O node. I/O node harus mempunyai sebuah direktori pada disk lokal untuk menyimpan file data.

Manager membuat, menghapus dan mengubah file file metadata yang disimpan dalam file system. File file ini mempunyai nama yang digunakan oleh aplikasi. File file ini bagaimanapun berukuran kecil dan hanya berisikan informasi tentang file permission dan lokasi data dan bukan data itu sendiri.

I/O daemon meletakkan file data dalam direktori bernama fXXXXXX.Y, dimana X merepresentasikan nomor inode dari file dan Y adalah indeks file yang sedang tidak digunakan. File file ini disimpan dengan hak hak yang diset sehingga hanya bisa dibaca oleh I/O daemon.

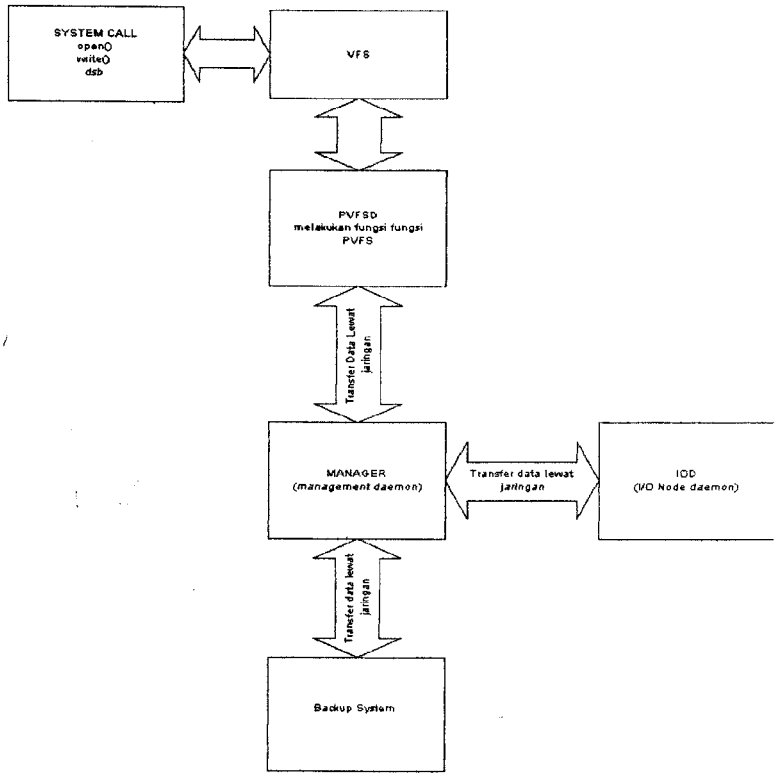
5.2.1 ARSITEKTUR PERANGKAT LUNAK

Seperti yang telah dituliskan, PVFS beroperasi bisa dengan menggunakan modul kernel dan juga I/O trap system library. Modul kernel akan mengakses I/O node melalui VFS (Virtual File System) Layer yang disediakan linux, sedangkan I/O trap akan melakukan proses-trap terhadap system library untuk menjalankan system call PVFS baru kemudian menjalankan library yang ada.

Penggunaan file system modul mempunyai arsitektur sebagai berikut dimana semua system call akan melalui virtual file system layer dahulu sebelum melakukan proses penulisan data pada I/O node.

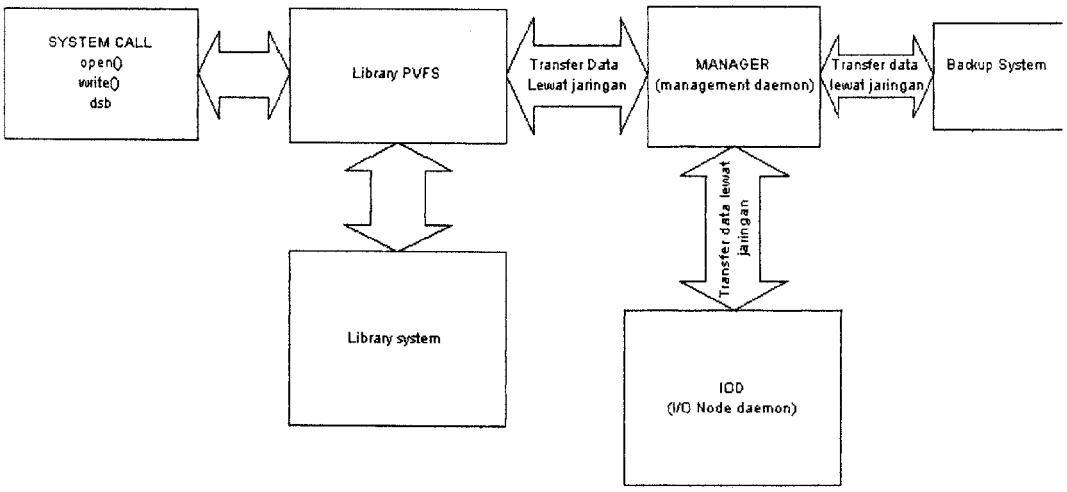
System call dalam gambar dibawah merupakan call pada subrutin yang biasanya dipanggil pada operasi file seperti open, putc dan sebagainya. VFS akan menggunakan rutin pada modul untuk melakukan operasi yang dimaksudkan. Pvfsd melakukan fungsi fungsi PVFS untuk melakukan akses data PVFS di manager dan I/O node.

Di lain sisi manager akan berinteraksi dengan I/O node dan system backup untuk melakukan operasi disk. Sistem backup akan dikirimi seiring dengan pengiriman data pada I/O node.



Gambar 5.2
Arsitektur PVFS dengan file system modul

Sedangkan pada I/O trapping VFS tidak diperlukan. I/O trap hanyalah menjalankan library pvfs dan kemudian menjalankan library yang sesungguhnya. library pvfs berisikan rutin rutin untuk berhubungan dengan manager dan i/o node. berikut ini adalah arsitektur perangkat lunak dengan menggunakan I/O trapping



Gambar 5.3
Arsitektur PVFS dengan I/O trappng

Pada I/O trapping, library standar linux di *–override* oleh library PVFS dengan variabel *environment* LD_PRELOAD hal ini mengakibatkan sebelum melakukan library standar akan melakukan operasi PVFS dulu. Operasi PVFS inilah nantinya yang akan melakukan interaksi dengan manager.

5.2.3 KONFIGURASI SOFTWARE

File file berikut ini dibuat oleh system administrator sebelum file system digunakan, file .pvfsdir menyimpan informasi tentang lokasi tempat manager untuk file system dan informasi tentang root direktori.

Sedangkan file .iodtab menyimpan daftar dari lokasi I/O daemon dan nomor port yang dipakai untuk system. Kedua file ini dibuat dengan menggunakan script mkiodtab

File .pvfsdir berbentuk teks file dan mempunyai format sebagai berikut :

Nomor Inode dari pvfsdir
Userid untuk direktori
Groupid untuk direktori
Hak akses untuk direktori
Nomor port manager
Nama mesin manager
Nama root direktori
Nama subdirektori

Tabel 5.1
Format file .pvfsdir

Berikut ini adalah contoh dari file .pvfsdir

```
116314
25
6000
0040775
3000
grendel
/pvfs
/
```

Akan ada file .pvfsdir di tiap direktori. Manager akan secara otomatis membuat file file tersebut ketika subdirektori dibuat.

File .iodtab juga dibuat oleh system administrator. Iodtab terdiri dari daftar host atau Alamat IP dan nomor port yang dipilih. .iodtab disimpan di root direktori dari file system PVFS. Sebuah contoh dari file .iodtab adalah sebagai berikut

```
192.168.0.1:7010
192.168.0.2:7010
192.168.0.3:7010
```

port 7010 merupakan nomor port yang dipilih untuk iod dan 192.168.0.x merupakan alamat IP dimana I/O daemon berada.

Iod akan mencari file konfigurasi yang bernama /etc/iod.conf ketika pertama kali dijalankan. File ini berisikan sejumlah parameter konfigurasi untuk I/O daemon termasuk direktori data, user dan group dimana I/O daemon berjalan dan nomor port untuk operasi I/O daemon.

Tiap baris terdiri dari dua field yaitu field selektor dan field untuk nilai. Dua field ini dipisahkan oleh satu spasi atau tab atau lebih. Format file iod.conf adalah sebagai berikut

Port <nomor_port>
user <userid>
group <groupid>
rootdir <direktori>
datadir <direktori>
logdir <direktorilog>

Tabel 5.2
Format file iod.conf

berikut ini adalah sebuah contoh dari file iod.conf

```
# IOD Configuration file, iod.conf
#
port 7001
user nobody
group nobody
rootdir /
datadir /pvfs_data
logdir /tmp
debug 0
```

I/O node bisa terletak pada satu mesin. Iod akan menggunakan file .iodtab dan beberapa file iod.conf. .iodtab terdapat dalam direktori metadata dan digunakan oleh manager untuk menentukan dimana lokasi iod. Peletakan I/O Daemon dalam satu mesin bisa dilakukan dengan cara menentukan port port yang berbeda dan konfig yang berbeda pada waktu menjalankan iod. Sebagai contoh adalah konfigurasi berikut ini yang akan meletakkan dua iod pada mesin yang sama

```
192.168.0.1:7000
192.168.0.1:7001
192.168.0.2:7000
192.168.0.2:7001
```

Konfigurasi .iodtab diatas menentukan bahwa empat iods akan digunakan untuk file system. Dua berjalan pada mesin dengan nomor IP address 192.168.0.1. iod yang satu berjalan pada port 7000 dan iod yang satunya akan berjalan pada port 7001. Seperti halnya pada mesin dengan nomor IP address 192.168.0.2

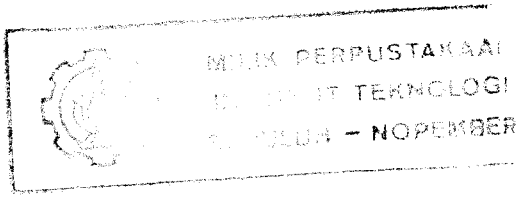
File-file iod.conf berikut ini merupakan konfigurasi untuk melakukan konfigurasi diatas.diasumsikan bahwa iod akan diletakkan pada direktori /pvfs_disk0 dan /pvfs_disk1.

Config file 1, "/etc/iod0.conf":

```
port 7000
user nobody
group nobody
rootdir /
datadir /pvfs_disk0
logdir /tmp
```

Config file 2, "/etc/iod1.conf":

```
port 7001
user nobody
group nobody
rootdir /
datadir /pvfs_disk1
logdir /tmp
```

5.3 HASIL UJICOBA

Ujicoba dilakukan dengan menjalankan software aplikasi Bonnie . Bonnie merupakan software aplikasi untuk mengukur kemampuan I/O dari suatu disk. Bonnie melakukan serangkaian ujicoba pada file yang ukurannya sudah ditentukan. Jika ukuran tidak ditentukan maka Bonnie menggunakan ukuran 100 Megabyte. Bonnie melakukan serangkaian ujicoba pada pvfs dengan melakukan penulisan, dan pembacaan serta pencarian secara random.

Pada ujicoba ini akan dibandingkan kecepatan I/O antara file system yang ada (second extended file system) dengan PVFS dengan n node pada waktu menuliskan file berukuran 5 megabyte

Berikut merupakan hasil ujicoba pada file system second extended file system di linux (tidak menggunakan pvfs) dengan besar file 5 MB

```
File '//Bonnie.8516', size: 5242880
Writing with putc()...done
Rewriting...done
Writing intelligently...done
Reading with getc()...done
Reading intelligently...done
Seeker 1...Seeker 3...Seeker 2...start 'em...done...done...done...
              ------Sequential Output-------- ---Sequential Input-- --Random--
              -Per Char- --Block--- -Rewrite-- -Per Char- --Block--- --Seeks---
Machine    MB K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU  /sec %CPU
            5  3745 51.2 67888 92.8  3646  7.8  3689 48.3 112052 65.7 8896.4 86.7
```

Kecepatan Penulisan	67888 K/sec
Kecepatan Membaca	112052 K/sec

Tabel 5.3
Hasil ujicoba I/O pada second extended file system

Hasil Kecepatan ini akan dibandingkan dengan kecepatan penulisan/pembacaan pada pvfs . Dimaksudkan dari hasil ujicoba ini dapat diketahui perbandingan kecepatan I/O antara paralel virtual file system dengan second extended file system.

Ujicoba 1

Jumlah i/o node 2 dan masing masing tersebar di dua mesin yang berbeda. Service i/o node dijalankan pada mesin 202.154.63.15 dan yang lain berjalan di 10.126.13.12 pada port 7000.

```
[root@cache2 bonnie]# ./Bonnie -d /mntpv -s 5
File '/mntpv/Bonnie.8506', size: 5242880
Writing with putc()...done
Rewriting...done
Writing intelligently...done
Reading with getc()...done
Reading intelligently...done
Seeker 1...Seeker 2...Seeker 3...start 'em...done...done...done...
              -------Sequential Output-------- ---Sequential Input-- --Random--
              -Per Char- --Block--- -Rewrite-- -Per Char- --Block--- --Seeks---
Machine    MB K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU  /sec %CPU
            5  1127 17.2   899  0.2   480  0.4  1259 17.2  1309  1.3  75.9  0.8
```

Kecepatan Penulisan	899 K/sec
Kecepatan Membaca	1309 K/sec

Tabel 5.4
Hasil ujicoba I/O pada jumlah node = 2

Dari hasil uji coba tersebut dapat dilihat bahwa kecepatan penulisan sama dengan 899 K/sec dan kecepatan membaca sebesar 1309 K/sec. Yang mana menunjukkan performance yang sangat rendah dibandingkan dengan file system second extended file system.

Ujicoba 2

Ujicoba berikut ini dijalankan dengan menggunakan 4 I/O node, dimana pada mesin 202.154.63.15 dijalankan dua I/O node dan pada mesin 10.126.13.12 dijalankan dua I/O node

```
[root@cache2 bonnie]# ./Bonnie -d /mntpv -s 5
File '/mntpv/Bonnie.8647', size: 5242880
Writing with putc()...done
Rewriting...done
Writing intelligently...done
Reading with getc()...done
Reading intelligently...done
Seeker 1...Seeker 2...Seeker 3...start 'em...done...done...done...
            -------Sequential Output-------- ---Sequential Input-- --Random--
            -Per Char- --Block--- -Rewrite-- -Per Char- --Block--- --Seeks---
Machine    MB K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU  /sec %CPU
           5  1128  9.5   795 -0.0   510  0.3  1047 13.3  1002  1.0  76.6  1.1
```

Kecepatan Penulisan	795 K/sec
Kecepatan Membaca	1002 K/sec

Tabel 5.5
Hasil ujicoba I/O pada jumlah node = 4

Dari hasil uji coba tersebut dapat dilihat bahwa kecepatan penulisan sama dengan 795 K/sec dan kecepatan membaca sebesar 1002 K/sec. Yang mana menunjukkan performance yang rendah dibandingkan dengan file system second extended file system.

Ujicoba 3

Ujicoba berikut ini dijalankan dengan menggunakan 6 I/O node, dimana pada mesin 202.154.63.15 dijalankan tiga I/O node dan pada mesin 10.126.13.12 dijalankan tiga I/O node

```
root@cache2 bonnie]# ./Bonnie -d /mntpv -s 5
File '/mntpv/Bonnie.8822', size: 5242880
Writing with putc()...done
Rewriting...done
Writing intelligently...done
Reading with getc()...done
Reading intelligently...done
Seeker 1...Seeker 2...Seeker 3...start 'em...done...done...done...
            -------Sequential Output-------- ---Sequential Input-- --Random--
            -Per Char- --Block--- -Rewrite-- -Per Char- --Block--- --Seeks---
Machine    MB K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU K/sec %CPU  /sec %CPU
           5  1124 16.3   836  0.3   497  0.4  1286 16.6  1372  0.8  76.1  0.9
```

Kecepatan Penulisan	836 K/sec
Kecepatan Membaca	1372 K/sec

Tabel 5.6
Hasil ujicoba I/O pada n=6

Dari hasil uji coba tersebut terlihat bahwa kecepatan penulisan sama dengan 836 K/sec dan kecepatan membaca sebesar 1372 K/sec. Dimana masih menunjukkan performance yang rendah dibandingkan dengan file system second extended file system.

Ternyata dari ketiga hasil ujicoba diatas performance dari I/O masih rendah. Hal ini bisa saja dikarenakan berbagai faktor, seperti kecepatan jaringan, beban server pada saat penulisan pada I/O nodes.

Beban server turut mempengaruhi dalam segi performance. jika beban tinggi, maka I/O node akan lebih lambat menulis ke disk. Konfigurasi ideal haruslah bahwa satu service i/o node diletakkan pada mesin tersendiri dan prosesnya tidak boleh terlalu banyak.

Kecepatan jaringan juga mempengaruhi dalam hal transfer data antara mesin. jika kecepatan rendah maka secara otomatis akan memperlambat kerja manager dalam menyusun ulang data data pada I/O node.

Pada hasil ujicoba yang dilakukan pada NASA Goddard Space Flight Center.yang menggunakan 64 mesin dengan konfigurasi sistem sebagai berikut

- Dual-processor Pentium Pro 200MHz, 128MB RAM
- 6 GB Seagate IDE drive

- 100Mbit Intel EtherExpress Pro in full duplex mode
- Sistem operasi REDHAT Linux 2.2.5

Ujicoba dengan menggunakan software Bonnie memperlihatkan 8.81MB/sec untuk proses penulisan dengan utilisasi 27.1% CPU dan 7.51MB/sec pada proses pembacaan dengan utilisasi 17.3% CPU.

64 mesin pada sistem dibagi menjadi 16 I/O node dan 48 manager untuk melakukan test tersebut. Test tersebut menunjukkan bahwa konfigurasi hardware juga menentukan dalam kecepatan I/O pada file system. Semakin bagus konfigurasi hardware semakin bagus pula unjuk kerja I/O pada paralel virtual file system.

BAB VI

KESIMPULAN DAN SARAN

Dari hasil ujicoba pada bab V, menunjukkan bahwa implementasi kinerja PVFS di jurusan teknik Informatika masih rendah. Hal ini disebabkan karena beberapa faktor seperti beban mesin yang menjalankan masing masing daemon server dan kecepatan jaringan. Beban mesin yang tinggi menjadikan proses penulisan oleh I/O node menjadi lebih lambat sehingga mengakibatkan manager dari PVFS juga menjadi lambat dalam menyusun kembali strip strip file pada I/O node.

Walaupun kinerja PVFS masih rendah, akan tetapi PVFS bisa dijadikan alternatif untuk penyimpanan file yang terdistribusi. hal ini bisa menambah ruang harddisk sebanyak jumlah mesin yang difungsikan sebagai I/O node.

Penerapan PVFS sebagai sistem terdistribusi juga dapat dilakukan. Hal ini dikarenakan pengguna dapat secara transparan dalam menggunakan PVFS, dapat dikatakan user tidak perlu mengetahui apakah suatu file system itu termasuk PVFS atau bukan. pengguna hanya menggunakan PVFS seperti halnya file system file system yang konvensional.

Sistem backup menghasilkan kemampuan redundansi. Sistem akan melakukan backup pada keadaan normal. manager melakukan backup dengan cara meletakkan data pada backup system seiring dengan penyimpanan data pada I/O node. Akan tetapi sistem backup ini

tidak bisa dikembalikan seperti keadaan sebelumnya, jika telah dipakai. Sistem backup akan terus digunakan setelah ada I/O node yang rusak. Proses rekoveri dilakukan dengan cara menyalin file data di backup node pada I/O node yang baru.

Saran saran yang dapat disampaikan pada tugas akhir ini adalah bahwa I/O node haruslah diletakkan di mesin dengan konfigurasi yang bagus dan cepat. untuk mendukung proses penulisan ke disk pada I/O node dan manager, dan juga haruslah dibuat sistem backup yang bisa rekoveri secara otomatis. dan tidak perlu menyalin dari sistem backup.

DAFTAR PUSTAKA

[Arg] Chiba City Project, the Argonne scalable cluster. *http://www.mcs.anl.gov/chiba/.*

[Bor-97] Rajesh Bordawekar, Steven Landherr, Don Capps, and Mark Davis. Experimental evaluation of the Hewlett-Packard Exemplar file system. *ACM SIGMETRICS Performance Evaluation Review*, 25(3):21-28, December 1997.

[Bra-98] Peter J. Braam. The Coda distributed file system. *Linux Journal*, #50, June 1998.

[Bra-98] Peter J. Braam, Michael Callahan, and Phil Schwan. The InterMezzo filesystem. In *Proceedings of the O'Reilly Perl Conference 3*, August 1999.

[Bra-00] Tim Bray. Bonnie file system benchmark. *http://www.textuality.com/bonnie/.*

[Car-00] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, ``PVFS: A Parallel File System For Linux Clusters", Preprint ANL/MCS-P804-0400, *submitted to the 2000 Extreme Linux Workshop*, April, 2000.

[Cod] Coda file system. *http://www.coda.cs.cmu.edu.*

[Int-95] Intel Scalable Systems Division. Paragon system user's guide. Order Number 312489-004, May 1995.

[Lig-96] W. B. Ligon III and R. B. Ross, ``Implementation and Performance of a Parallel File System for High Performance Distributed Applications", *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, Agustus, 1996.

[Lig-99] W. B. Ligon III and R. B. Ross, ``An Overview of the Parallel Virtual File System" *Proceedings of the 1999 Extreme Linux Workshop*, Juni, 1999.

[Ste-92] W. Richard Stevens, "Advanced Programming in The UNIX Environment", Addison-Wesley Professional Computing Series,1992

[sgi-00] XFS: A next generation journalled 64-bit filesystem with guaranteed rate I/O. http://www.sgi.com/Technology/xfs-whitepaper.html.