

ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA
SEDE DI CESENA
FACOLTA' DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA IN SCIENZE DELL'INFORMAZIONE

TITOLO DELLA TESI

**PROGETTAZIONE, REALIZZAZIONE ED
OTTIMIZZAZIONE DI UN
CLUSTER IBRIDO 32/64 BIT
PER HPC ALTAMENTE AFFIDABILE**

Tesi di laurea in

Fisica Numerica

Relatore
Chiar.mo Prof. Renato Campanini

Presentata da
Christian Zoffoli

Correlatore
Dott. Matteo Roffilli

Sessione I
Anno Accademico 2004/2005

*Ai miei genitori che mi hanno sempre
incoraggiato e sostenuto.*

*A Cristina che mi è sempre stata
vicina nei momenti difficili.*

INDICE

INTRODUZIONE	I
---------------------------	----------

CAPITOLO 1

INTRODUZIONE ALL'HPC	1
1.1 Architetture	2
1.2 Sistemi Operativi tipici dei cluster HPC	4
1.3 High Performance Computing e High Availability	6
1.4 Necessità dei sistemi HPC ad alta disponibilità in generale.....	11
1.5 Necessità dei sistemi HPC ad alta disponibilità nello specifico per applicazioni di imaging medico	12
1.6 HPC ed architetture parallele	13
1.7 Cluster Beowulf	14
1.8 Classificazione dei Cluster Beowulf	17
1.9 HA come estensione di un cluster HPC	18

CAPITOLO 2

HPC A 32 BIT	20
2.1 Progettazione	20
2.2 Hardware adottato	22
2.3 Scelta del sistema operativo	23
2.4 Installazione del sistema base	25
2.5 Creazione dell'infrastruttura di rete	35
2.6 Configurazione dei servizi DHCP, PXE e NFS per i nodi diskless	37
2.7 Realizzazione del prototipo della Single System Image e realizzazione di un middle-layer per l'autoconfigurazione dei nodi slave HPC	39
2.8 Risoluzione dei nomi	42
2.9 Installazione del servizio Secure Shell in modalita' shared key	43
2.10 Configurazione delle autenticazioni centralizzate LDAP	43

2.11 Installazione e configurazione di DISTCC e CCACHE per ridurre i tempi di sviluppo	45
2.12 Installazione e configurazione di MPI	46
2.13 Risoluzione delle problematiche di sicurezza del Cluster	48

CAPITOLO 3

HPC A 64 BIT (AMD64 OPTERON)	49
3.1 Introduzione alla tecnologia AMD64	49
3.2 Analisi dell'hardware	50
3.3 Configurazione ed Installazione di un kernel a 64bit	53
3.4 Adattamento delle distribuzioni linux alla nuova architettura AMD64	54
3.5 Ottimizzazione e test prestazionali su compilatori di nuova generazione (GCC-3.4.x)	55

CAPITOLO 4

HA	56
4.1 Livelli di disponibilità	57
4.2 Il requisito dei cinque 9	61
4.3 Due Geometrie di clustering HA: Active-Active ed Active-Standby	65
4.4 Creazione di master ridondanti	66
4.5 Abilitazione del supporto LVS nei due nodi master HPC	66
4.6 Installazione e configurazione di Heartbeat per la gestione delle problematiche di HA all'interno del Cluster	67
4.7 Sincronizzazione delle mappe IPVS tra i directors	69
4.8 Adattamento dei servizi di rete al supporto di HA	70
4.9 Aggiunta del supporto failover a DHCP	71
4.10 Abilitazione delle repliche in OpenLDAP	72
4.11 Replicazione del servizio di risoluzione dei nomi	72
4.12 Analisi delle soluzioni per la replicazione dei dati tra i due nodi master	73
4.13 Distributed Filesystem: Coda	75
4.14 Clustered filesystem: OGFS, RedHat GFS, Oracle OCFS	75

4.15 Distributed Replicated Block Device: DRBD	77
4.16 Installazione e configurazione di DRBD ed integrazione con heartbeat	78
4.17 Adattamento del middle-layer alle nuove funzionalita' di HA	79
4.18 Attivazione delle funzionalita' di load balancing: round robin e weighted round robin	80

CAPITOLO 5

TEST	82
5.1 Compilazione distribuita, analisi delle performance	82
5.2 Analisi delle performance di DRBD paragonato ad un block device locale	84
5.3 Analisi dei tempi medi di transizione delle risorse da un director attivo ad uno operante in modalit� passiva	88

CONCLUSIONI	III
--------------------------	------------

BIBLIOGRAFIA	V
---------------------------	----------

APPENDICE	IX
------------------------	-----------

INTRODUZIONE

Il presente lavoro di tesi è stato svolto in collaborazione con un gruppo di ricercatori dell'università degli Studi di Bologna costituito da ricercatori del Corso di Laurea di Scienze dell'Informazione (sede di Cesena) e del Dipartimento di Fisica. Tale gruppo è stato costituito per elaborare un programma CAD (Computer Aided Detection) capace di individuare le masse tumorali in immagini generate da apparecchiature radiodiagnostiche durante esami mammografici.

Il lavoro di tesi ha avuto come scopo la creazione di un cluster High Performance Computing (HPC) dotato di funzionalità di High Availability (HA).

Tale realizzazione si è resa necessaria per meglio supportare in termini di continuità del servizio il lavoro dei radiologi che devono diagnosticare con la massima celerità eventuali tumori al seno.

Il sistema distribuito in oggetto è stato costruito utilizzando due macchine a basse performance, basate su CPU x86 di tipo Intel Pentium 3, per la realizzazione dei nodi master HPC, dotati di funzionalità ad alta disponibilità, e da tre nodi, basati su CPU x86 AMD Athlon MP, per la realizzazione dei nodi di calcolo del cluster HPC.

Tutti i nodi del cluster sono dotati di sistema operativo Gentoo Linux e sono interconnessi mediante schede di rete di tipo ethernet. I nodi master HPC, operanti in alta disponibilità, sono dotati di sottosistema a dischi parzialmente replicati attraverso un sistema per la replicazione distribuita dei block devices, denominato DRBD, mentre i nodi di calcolo HPC sono completamente privi di dischi e pertanto caricano ogni dato direttamente dal Network File System (NFS) esportato dai nodi master HPC.

Il centro del sistema ad alta disponibilità è costituito dalla suite Heartbeat, alla quale è demandata la gestione delle risorse del cluster e che è in grado di monitorare e reagire ad eventuali malfunzionamenti del nodo master HPC che sta erogando il servizio.

Nel capitolo 1, sono state presentate le basi teoriche riguardanti i sistemi distribuiti, i diversi modelli teorici e le diverse geometrie di cluster. In particolare si è posto l'accento sulle differenze tra un cluster di tipo Beowulf ed un cluster creato per l'alta disponibilità HA.

Nel capitolo 2, si è analizzata l'installazione e la configurazione della distribuzione Gentoo Linux sui nodi master e slave HPC operanti a 32bit.

Nel capitolo 3, si è analizzata l'installazione di una distribuzione Gentoo Linux a 64bit su una macchina basata su tecnologia AMD 64 Opteron.

Nel capitolo 4, si sono analizzati i vari aspetti tecnico-implementativi per l'integrazione di funzionalità ad alta disponibilità in cluster per il calcolo ad alte performance HPC.

Nel capitolo 5, si sono analizzati i dati prodotti attraverso dei test sull'intero sistema.

Chiude il lavoro di tesi un'appendice contenente alcuni file testuali di configurazione di diversi software: Sistema operativo, MPI, DRBD, Heartbeat, DHCP, DNS, OpenLDAP, ecc.

CAPITOLO 1

INTRODUZIONE ALL'HPC

Pur essendo una scienza relativamente giovane, l'informatica ha invaso ogni strato della società civile. L'invenzione di macchine capaci di eseguire automaticamente operazioni complesse in lassi di tempo molto ridotti ha permesso di addentrarsi in realtà prima d'ora inesplorate. In molti settori legati alla ricerca ed all'analisi di realtà complesse (analisi scientifica, finanziaria ecc.), inizialmente si sono diffusi prodotti informatici specializzati e realizzati appositamente allo scopo.

Con l'avvento dell'informatica personale e con la creazione di ciò che può esserne ritenuto il simbolo, il personal computer, realtà consolidate quali quella dei supercomputer hanno iniziato ad accusare i primi colpi. La richiesta di ingenti capacità di elaborazione a costi bassi ha spinto verso la creazione di software in grado “virtualizzare” un gruppo di macchine creando un unico sistema in grado di distribuire il carico operativo su ogni componente del gruppo.

Più computer non specializzati aggregati, visibili agli occhi dell'utente come un unico computer virtuale, focalizzati nella risoluzione di un problema comune, vengono definiti sistema distribuito o “cluster”¹.

Il paradigma alla base di un sistema distribuito è assimilabile alla teoria “divide et impera”, secondo la quale ogni problema può essere risolto scomponendolo in sottoproblemi più piccoli, la soluzione di ognuno dei quali genera una parte della soluzione del problema iniziale. L'applicazione di questo schema di ragionamento

¹Un cluster (dall'inglese *grappolo*) è un insieme di computer connessi tramite una rete telematica. Lo scopo di un cluster è quello di distribuire una computazione molto complessa tra i vari computer componenti il cluster. In sostanza un problema che richiede molte elaborazioni per essere risolto viene scomposto in sottoproblemi separati i quali vengono risolti in parallelo. Questo ovviamente aumenta la potenza di calcolo del sistema.

incorpora il concetto di modularità che caratterizza tutti i moderni sistemi di calcolo, in modo più evidente i sistemi distribuiti.

Un sistema distribuito, infatti, non è altro che un insieme di calcolatori identici che si definiscono “nodi”. In un sistema distribuito si ha la possibilità di aggiungere o eliminare i nodi generando rispettivamente un aumento oppure una diminuzione della potenza di calcolo totale dell'intero sistema. Questa caratteristica prende il nome di scalabilità. Un sistema si definisce scalabile quando l'aumento delle prestazioni di calcolo aumenta proporzionalmente con il numero di nodi del sistema.

Secondo la definizione tipica, un cluster è un sistema distribuito composto da un numero arbitrario di singoli computer, definiti nodi. I nodi, interconnessi tra loro da una rete di comunicazione privata che viene usata esclusivamente per la sincronizzazione dei dati e l'interscambio dei messaggi fra i processi in esecuzione, condividono le risorse che possono essere sia locali che remote.

1.1 Architetture

A seconda della tipologia di gestione e della collocazione delle risorse, un sistema distribuito può essere di due differenti tipi:

1. “Tightly coupled”
2. “Loosely Coupled”

1. Nei sistemi denominati “Tightly coupled”, detti anche “Shared Memory”, abbiamo un insieme di nodi che condividono un'unica porzione di memoria, utilizzata anche per scopi di sincronizzazione e comunicazione.

2. Nei sistemi “Loosely Coupled” abbiamo un insieme di nodi dotati di memoria propria che scambiano informazioni attraverso un'interconnessione fisica ed una tecnica denominata “Message Passing”.

Il sistema che andremo ad analizzare nel dettaglio, corrisponde alla definizione di sistema distribuito “Loosely Coupled”.

All'interno dei due tipi di sistemi distribuiti appena illustrati possiamo individuare dei sottoinsiemi che meglio definiscono la tipologia di soluzione adottata.

Fanno parte della categoria “Tightly Coupled” i sistemi definibili come minicomputer che in realtà non sono altro che sistemi dotati di un certo numero di processori che condividono le stesse risorse centralizzate mantenendo al contempo caratteristiche di scalabilità e modularità. La sincronizzazione delle operazioni e lo scambio di messaggi tra di essi avviene utilizzando aree della memoria principale ed ogni operazione d'inserimento dati e consultazione viene realizzata utilizzando appositi terminali oppure workstation.

Nella categoria “Loosely Coupled” possiamo individuare i seguenti modelli:

1. workstation model;
2. workstation-server model;
3. processor pool model.

1. I sistemi definibili come “workstation model” sono caratterizzati dalla presenza di un insieme di workstation che gestiscono autonomamente le proprie risorse e che sono dotate sia di un sistema di gestione dei processi locali, sia di un sistema per la gestione dei processi remoti in grado di spostare i processi sui nodi meno carichi.

2. I sistemi denominati “workstation-server model” sono caratterizzati dalla flessibilità del modello “workstation model”, dato che anch'essi sono costituiti da un gruppo di workstation interconnesse, ma ereditano anche i vantaggi del modello a minicomputer. Tale soluzione permette l'esecuzione di operazioni generiche sulle workstation e di operazioni specializzate sui minicomputer dedicati.

I vantaggi forniti da questo modello possono essere di importanza rilevante, quando sono richiesti livelli di performance molto elevati, e pertanto in molti casi la scelta risulta essere obbligata.

Grazie a questo modello, per esempio, è possibile centralizzare le risorse di memorizzazione di massa, costituendo appositamente un minicomputer come file server, in modo da fornire prestazioni eccellenti nell'input-output dei dati, semplificare la manutenzione ed evitare l'overhead della migrazione dei processi dati, ed è possibile inoltre eseguire elaborazioni remote attraverso i protocolli client-server.

3. Il modello “processor pool” è caratterizzato dalla presenza di nodi generici non specializzati che vengono dinamicamente allocati dal sistema di gestione delle risorse e che vengono rappresentati esternamente con l'immagine di un sistema singolo. L'utente che deve utilizzare le risorse di calcolo del cluster non si collega ai nodi, ma a terminali specializzati che non fanno parte del cluster. Pertanto, la percezione che ne risulta è quella di collegarsi all'intero sistema.

Per la sua praticità ed efficacia, questo modello risulta essere quello più diffuso tra gli attuali cluster adibiti al calcolo massivo.

1.2 Sistemi Operativi tipici dei cluster HPC

Il sistema operativo di un sistema distribuito deve essere tale da consentire e favorire le operazioni tipiche di un cluster. Allo stato attuale vengono individuate due tipologie di sistemi operativi utilizzati all'interno di cluster HPC:

1. “Network Operating System”;
2. “Distributed Operating System”.

La differenza principale tra queste due tipologie risiede principalmente nel tipo di visione del sistema che il sistema stesso è in grado di dare all'utente.

1. Nei sistemi denominati “Network Operating System” l'utente ha un percezione precisa dei nodi, delle risorse occupate e può scegliere dove avviare la propria elaborazione.
2. Nei sistemi denominati “Distributed Operating System” l'utente ha solo la percezione d'insieme e sarà il sistema stesso a determinare dove e come verrà eseguita la computazione.

Da un punto di vista progettuale i sistemi operativi di tipo “Distributed Operating System” hanno un livello di complessità decisamente più marcato rispetto ai “Network Operating System”. Il sistema stesso, infatti, deve gestire problematiche legate ai guasti, alla ridondanza dei componenti critici, alla sicurezza ed alle performance, mantenendo livelli di performance e scalabilità elevati. Nei sistemi operativi di tipo “Network Operating System” è l'utente stesso, in particolare l'amministratore, che deve intervenire in caso di guasto di parte del sistema ed è altresì l'utente che è demandato a decidere su quale nodo avviare l'elaborazione delle informazioni, oltre al fatto che la maggior parte dei meccanismi che rendono il sistema operativo non sono celati dal sistema stesso e pertanto la complessità d'insieme risulta notevolmente ridotta.

1.3 High Performance Computing e High Availability

Una volta definite le tipologie principali di sistemi distribuiti, risulta doverosa un'altra distinzione logica che riguarda propriamente l'utilizzo a cui il cluster è dedicato.

Un cluster può essere di tipo HPC (High Performance Computer) oppure di tipo HA (High Availability).

I cluster di tipo HPC sono normalmente dedicati al calcolo massivo, peculiare delle simulazioni scientifiche e finanziarie in più variabili. Normalmente i nodi sono composti da un insieme di personal computer che, gestendo autonomamente le proprie risorse, sono assimilabili al modello “Loosely Coupled” precedentemente analizzato.

Al fine di ottimizzare i costi e ridurre il numero di componenti soggetti a rottura, i nodi vengono privati di tutte quelle periferiche, di uso comune, assolutamente non necessarie alle applicazioni di supercalcolo. Pertanto vengono eliminati componenti quali: tastiere, mouse, monitor, schede grafiche, schede audio, floppy, cdrom ed in alcuni casi anche l'hard disk (ad esempio nei cluster diskless). Al contempo i nodi sono molto curati dal punto di vista dei processori, della memoria centrale e della connettività. I processori adottati, generalmente due o quattro, sono coadiuvati da ingenti quantitativi di ram (due o più Gigabyte) a bassa latenza e da chipset² capaci di sopportare elevati flussi di dati. Inoltre, vengono investiti notevoli percentuali del budget nell'acquisto di periferiche di rete velocissime e dotate di tempi di risposta molto bassi. Allo stato attuale non sono rari cluster con schede di rete capaci di velocità di trasferimento nell'ordine di qualche Gigabit. Per quanto riguarda le connessioni di rete, in moltissime applicazioni di supercalcolo risulta essenziale ridurre al minimo la latenza media delle comunicazioni tra i nodi, pertanto, risultano sempre più frequenti gli investimenti in tecnologie non propriamente destinate al mercato consumer (ad esempio gli apparati Myrinet prodotti

² Circuito integrato che gestisce la comunicazione tra le varie componenti di una scheda

dall'americana Myricom³). Tale scelta, che sostanzialmente contrasta con il modello “processor pool” poiché vengono utilizzati prodotti non disponibili attraverso la grande distribuzione, spesso permette di ridurre i drastici cali di performance in sistemi dotati di un considerevole numero di nodi che devono garantire tempi di risposta estremamente ridotti. Abbassando i tempi necessari allo scambio dei dati e soprattutto allo scambio dei messaggi tra i nodi, otteniamo un sistema più snello e reattivo che non viene troppo penalizzato dall'adozione di nodi provenienti dal mercato consumer.

I cluster di tipo HA hanno come obiettivo primario quello di garantire la continuità di servizi con alti livelli di criticità.

Da ricerche quali quella condotta da IEEE nell'Aprile del 1995 emerge che una larga percentuale (oltre il 30%) dei disservizi all'intero delle reti locali è determinata dalla rottura di componenti hardware oppure da fattori umani imprevisti, che possono andare dalla semplice disconnessione del cavo di alimentazione o di rete ad un errata modifica della configurazione, all'accidentale spegnimento della macchina stessa.

Al fine di cercare una soluzione ad un numero sempre crescente di disservizi dovuti a cause impreviste, si cerca di schematizzare le tipologie di guasti e di aggiungere livelli sempre più sofisticati di ridondanza in grado di assorbire l'eventuale disservizio senza che l'utente percepisca il guasto.

Il primo livello d'intervento porta alla realizzazione di server facenti largo uso della ridondanza in reparti quali l'alimentazione, le connessioni di rete, la memoria centrale ed i sottosistemi a disco (generalmente tale soluzione è coadiuvata da un sistema di monitoraggio interno in grado di segnalare l'imminenza di un guasto in una determinata sezione del sistema).

Un secondo livello di ottimizzazione riguarda la creazione di cluster di due macchine in grado di migliorare il livello di affidabilità di un sistema, eliminando quello che viene definito il “single point of failure”, ovvero il punto, che se compromesso, pregiudica

³[Http://www.myri.com](http://www.myri.com)

l'erogazione del servizio (ad esempio le connessioni di rete non ridondati possono determinare l'insorgere di un “single point of failure”).

Allo stato attuale esistono due tipi di cluster HA:

1. quelli operanti in modalità attiva-passiva (detta anche “active-standby”);
2. quelli operanti in modalità attiva-attiva.

1. I cluster in modalità active-standby sono caratterizzati dalla presenza di un nodo primario che eroga il servizio e da un nodo secondario “dormiente” che prende il posto del primario in caso di guasto.
2. I cluster in modalità active-active sono caratterizzati dalla presenza di due nodi attivi che si scambiano le informazioni e che si dividono il carico delle richieste provenienti dall'esterno in modo del tutto automatico.

Allo stato attuale della tecnologia solamente le soluzioni active-standby risultano affidabili e largamente utilizzate, le implementazioni active-active presentano ancora comportamenti anomali che ne pregiudicano l'adozione su larga scala.

La creazione di coppie di server HA, che concettualmente può sembrare semplice, in realtà nasconde problematiche che richiedono riflessioni attente e settaggi minuziosi.

Prima di tutto, dobbiamo ricordare che, se si utilizza un risorsa ed il server che la eroga smette di funzionare correttamente, avremo un momento di vuoto nel quale il servizio non verrà erogato regolarmente. La corretta gestione di tale frangente risulta essere cruciale per evitare che l'utente percepisca il guasto. In linea di principio possiamo individuare tre fasi operative che caratterizzano la transizione dal server primario al secondario:

- fase 1: il server primario smette di erogare il servizio a causa della rottura di un componente chiave del sistema e segnala l'eventuale anomalia al server secondario oppure smette di inviare le informazioni di stato.
- fase 2: il server secondario riceve la segnalazione dell'anomalia oppure smette di ricevere le informazioni di stato ed inizia ad eseguire una serie di verifiche prima di assumere il controllo dei servizi.
- fase 3: il server superstite, avendo verificato la reale rottura del server attivo, inizia ad erogare il pacchetto dei servizi cercando di mantenere le connessioni attive, minimizzando il disservizio.

L'esecuzione di queste tre fasi è estremamente delicata e non esente da possibili effetti collaterali. Essenzialmente possiamo dividere le possibili anomalie in due categorie:

- la prima riguarda la chiusura delle eventuali connessioni attive, causa la migrazione del servizio da un server ad un altro.

A volte, nel caso di servizi stateless, ad esempio NFS⁴, HTTP/HTTPS⁵ ecc., è possibile ridurre quasi a zero la percezione della migrazione dell'erogazione di un servizio da un server ad un altro semplicemente adottando alcune accortezze in fase di configurazione oppure in caso di avvio del servizio stesso. Ad esempio, possiamo dire al demone⁶ che gestisce il servizio di rileggere eventuali informazioni riguardanti le connessioni attive, come i file di lock⁷, opportunamente registrate in una risorsa di rete condivisa, prima di riprendere l'erogazione del servizio stesso.

Nel caso di servizi con protocolli stateful, a meno che il servizio non supporti la clusterizzazione, è impossibile avere una migrazione indolore da un server ad un

⁴ NFS: Network File-System. Un server NFS esporta una certa quantità di spazio disco locale per renderlo accessibile a tutti gli host presenti sulla rete i quali possono montarlo come un device locale.

⁵ HTTP è l'acronimo di HyperText Transfer Protocol (protocollo di trasferimento di un ipertesto). Usato come principale sistema per la trasmissione di informazioni sul web.

⁶ Demone: in ambiente Linux un demone è un programma residente in memoria demandato all'erogazione di un determinato servizio. Ad esempio un demone NFS è in ascolto su una determinata porta per poter "servire" file e directory ai client autorizzati che ne facciano richiesta.

⁷ File di lock: file utilizzato per rendere disponibile in modo esclusivo una risorsa.

altro senza generare un temporaneo disservizio. Tale anomalia è determinata dal fatto che prima del guasto l'utente scambia le informazioni con un server, che successivamente alla rottura viene sostituito dal suo alter ego precedentemente in standby. Ovviamente, il nuovo server ed in particolar modo il servizio che deve erogare, se non predisposto alla clusterizzazione, non è a conoscenza delle operazioni in corso e pertanto si dovrà procedere con la riautenticazione del servizio, ove previsto, con la rinegoziazione delle risorse, ecc.

- La seconda anomalia, ben più grave, riguarda la situazione nella quale s'interrompe il flusso delle informazioni di stato tra le due macchine, vuoi per una rottura hardware, vuoi per un'anomalia software. In tal caso, entrambe le macchine, progettate per lavorare in modalità active-standby, tenderanno di erogare contemporaneamente il servizio. Tale condizione prende il nome di “brain splitting” e, nel caso in cui non vengano prese precauzioni, può portare alla corruzione/inconsistenza dei dati ed a comportamenti anomali nei sistemi che cercano di utilizzare il servizio.

Tipicamente si cerca di combattere questa eventualità utilizzando dei sistemi di gestione dell'alimentazione, chiamati stonith, in grado di interrompere l'erogazione dell'energia elettrica al nodo identificato come malfunzionante. Tale tutela permette di avere sempre e solo una macchina attiva allo stesso tempo e di non pregiudicare l'erogazione del servizio. Questo tipo di tecnologia spesso risulta essere integrata all'interno di distributori di alimentazione ed a gruppi di continuità e normalmente viene amministrata attraverso connessioni di rete (ethernet) oppure connessioni seriali.

Un'altra differenza saliente tra i cluster HPC ed i cluster HA, come risulta chiaro dall'analisi precedente, riguarda il numero di nodi coinvolti. Mentre nel caso di cluster HPC abbiamo decine di nodi, nel caso di cluster HA abbiamo tipicamente due soli nodi. Tale differenza normalmente è dettata dal fatto che un cluster HA ha dei carichi

computazionali molto ridotti rispetto ad un cluster HPC e dal fatto che la gestione degli effetti collaterali risulta molto complessa con un numero di nodi elevato.

Anche nel caso dei cluster HA l'utente ha una visione unica del sistema che eroga i servizi e la transizione delle risorse da un nodo all'altro avviene in modo del tutto automatico.

1.4 Necessità dei sistemi HPC ad alta disponibilità in generale

Negli anni novanta le due tipologie illustrate, HPC ed HA, viaggiavano su binari paralleli ed erano relegate in ambiti completamente differenti. Sul finire del secolo, la diffusione del calcolo ad alta performance anche in ambiti commerciali e soprattutto la diffusione sempre crescente di cluster realizzati utilizzando componenti disponibili attraverso la grande distribuzione (COTS: Commodity Off The Shelf), che tuttavia risultano generalmente meno affidabili, hanno determinato la necessità di fondere le funzionalità offerte da entrambi i gruppi in un'unica soluzione più complessa ma al contempo più flessibile ed egualmente affidabile.

Tipicamente, quando si sceglie di attivare porzioni ad alta disponibilità all'interno di cluster HPC si cerca d'intervenire in reparti quali: la condivisione di files, i database, i gateway, i firewall, le autenticazioni centralizzate oppure i sistemi di gestione dei lock di filesystem clusterizzati quali, ad esempio, GFS.

Pertanto, è chiaro che se l'analisi dei punti deboli del sistema risulta essere minuziosa si otterrà un cluster HPC-HA del tutto paragonabile ai sistemi operativi distribuiti più sofisticati.

1.5 Necessità dei sistemi HPC ad alta disponibilità nello specifico per applicazioni di imaging medico

Il presente lavoro è dedicato completamente alla creazione di una soluzione HPC ad alta disponibilità in grado di far girare un'applicazione CAD (Computer Aided Detection) distribuita per l'analisi delle immagini mammografiche all'interno di un contesto ospedaliero.

Per esplicita richiesta progettuale, il cluster, che può essere composto da 2 nodi HA e da N nodi HPC, deve tollerare la rottura di almeno 1 nodo HA ed N-1 nodi HPC senza generare un disservizio percepibile dall'utilizzatore. Inoltre il tempo di elaborazione di un set d'immagini deve essere di pochi minuti in modo da permettere a chi esegue l'esame di avere immediatamente un riscontro e pertanto intervenire tempestivamente nel caso vengano rilevate anomalie.

Pur essendo disponibili diversi pacchetti commerciali che integrano gli strumenti necessari per realizzare cluster HPC ed HA in fase di progettazione, si è scelto di utilizzare del software open source (libero e gratuito) funzionante sotto GNU/Linux⁸.

La scelta è dettata sia da questioni economiche, dato che i pacchetti commerciali sono spesso molto costosi, sia dal fatto che questi ultimi risultano essere meno mantenibili e personalizzabili, dato che i codici sorgenti non sono disponibili ed ogni modifica necessita dell'intervento del personale specializzato della casa madre.

⁸Degna di nota è la differenza tra la dicitura Linux e GNU/Linux: la prima sta ad indicare solamente il kernel del sistema operativo, originariamente sviluppato da Linus Torvalds, la seconda sta ad indicare l'insieme del kernel Linux e di svariati programmi open source sviluppati in seno al progetto GNU il cui padre è Richard Stallman.

Spesso questo insieme di componenti software più o meno esteso può essere chiamato "distribuzione linux" e può sia essere distribuito gratuitamente, sia messo in vendita abbinato a contratti di supporto. Molto spesso, erroneamente, si utilizza erroneamente il termine "linux" per riferirsi all'intera distribuzione, quando invece sta ad indicare solo una componente, ovvero il kernel.

1.6 HPC ed architetture parallele

Un sistema HPC identifica tipicamente un sistema in cui le risorse di calcolo sono particolarmente performanti. Tradizionalmente, i sistemi HPC erano rappresentati da macchine monolitiche capaci di alloggiare più CPU e con canali per l'interscambio dei dati estremamente performanti. Negli ultimi anni la corsa alla riduzione dei prezzi, la proliferazione dell'informatica di consumo e la creazione di software in grado di creare cluster virtuali utilizzando comunissimi personal computer ha permesso di creare due nuove tipologie di cluster che prendono il nome di Beowulf Cluster e Mosix Cluster.

La prima tipologia, altamente diffusa in ambiti scientifici, risulta essere la più collaudata e ricca di successi, mentre la seconda, caratterizzata da un susseguirsi di successi e disfatte, sta vivendo un periodo di stasi totale dello sviluppo.

L'approccio delle due soluzioni è nettamente contrastante e ne determina anche i pro ed i contro.

Per quanto riguarda i cluster Beowulf, il fatto che questi necessitino di un set di librerie creato appositamente (librerie PVM, cioè Parallel Virtual Machine, e/o MPI, cioè Message Passing Interface) per effettuare l'elaborazione distribuita su un gruppo di macchine potrebbe essere un problema insormontabile, nel caso si faccia uso di applicazioni commerciali delle quali non si dispone del codice sorgente. Al contempo si ha il notevole vantaggio che una volta che la nostra applicazione verrà scritta oppure riscritta per utilizzare le librerie PVM e MPI, avrà un incremento di performance molto elevato ed in alcuni casi quasi lineare con la crescita del numero dei nodi coinvolti nell'elaborazione.

Per quanto riguarda i cluster di tipo Mosix abbiamo il vantaggio di non dover riscrivere il codice, dato che il sistema è in grado di distribuire dinamicamente i processi sui nodi meno carichi del cluster, tuttavia, abbiamo il grosso svantaggio che non tutte le applicazioni possono migrare (ad esempio le applicazioni che fanno uso della shared memory non possono migrare) e spesso anche le applicazioni in grado di migrare non ottengono, per la loro conformazione, reali vantaggi dalla migrazione sui nodi remoti. Se ad esempio prendiamo il caso di applicazioni che utilizzano tanti processi dotati di vita brevissima, vediamo immediatamente come migrare il processo su un nodo remoto sia più costoso, in termini di risorse e di tempo di elaborazione, rispetto all'eseguire la stessa elaborazione in locale. Facendo tale riflessione dobbiamo sempre ricordare che le periferiche di rete, anche se veloci, sono sempre molto più lente rispetto ai canali di comunicazione interni alla macchina stessa. Pertanto risulta essere più vantaggioso migrare un processo che necessita di un tempo di elaborazione mediamente lungo rispetto alla migrazione di un processo dalla vita breve.

Nel caso dell'applicativo CAD utilizzato all'interno del cluster che viene analizzato in questa tesi sono state utilizzate le librerie MPI che forniscono una buona stabilità operativa, performance elevate e un supporto nativo dell'elaborazione distribuita.

Pertanto verrà approfondita unicamente la tipologia di clustering Beowulf e verrà tralasciata l'analisi dettagliata delle soluzioni basate su Mosix.

1.7 Cluster Beowulf

Esistono varie definizioni di cluster Beowulf. La definizione in senso stretto viene utilizzata per definire i cluster realizzati allo stesso modo del sistema originario⁹ della

⁹ Cluster Beowulf originario: il primo sistema costruito da Tom Sterling e Don Becker presso i laboratori della NASA negli anni '90.

NASA mentre una accezione più ampia della stessa identifica come cluster Beowulf tutti i sistemi workstation in grado di far girare del codice parallelo.

In questa tesi, verrà utilizzata una definizione di cluster Beowulf che si inserisce in una posizione intermedia tra le due precedentemente illustrate.

Consideriamo Beowulf un'architettura a multicomputer che può essere utilizzata per eseguire calcoli paralleli. E' un sistema normalmente composto da un nodo server e da uno o più nodi client connessi tra loro utilizzando connessioni Ethernet. E' un sistema costituito utilizzando componenti hardware comuni e facilmente reperibili, come qualunque PC che possa far girare il sistema operativo GNU/Linux, normali adattatori di rete Ethernet (oppure altre tipologie di rete) ed apparati di commutazione di rete di tipo switch¹⁰. Normalmente non contiene alcun componente hardware speciale ed è facilmente realizzabile.

Beowulf utilizza, inoltre, software comune come il sistema operativo GNU/Linux, Parallel Virtual Machine (PVM) e Message Parsing Interface (MPI). Ad un nodo server, che fornisce ogni tipo di risorsa ai nodi client, è demandato il controllo completo di tutto il cluster. Ad esso è demandato anche il ruolo di console di sistema e di punto di collegamento (gateway) con il mondo circostante. Nel caso di sistemi Beowulf di notevoli dimensioni possiamo avere anche più di un nodo server e possibilmente altri nodi dedicati ad assolvere compiti particolari come ad esempio console o stazioni di monitoraggio dello stato dell'intero cluster.

Nella stragrande maggioranza dei casi i nodi client presenti all'interno del sistema Beowulf sono dedicati unicamente all'elaborazione massiccia dei dati. I nodi vengono controllati e configurati in modo del tutto automatico dal nodo server ed eseguono unicamente le operazioni che gli vengono ordinate in remoto dal nodo server.

¹⁰ Switch: dispositivo hardware che consente il collegamento in rete di computer.

In configurazioni prive di unità di memorizzazione di massa denominate diskless, i nodi client, non conoscono nemmeno il loro indirizzo di rete (IP) e neppure il loro nome, fino a quando il server non provvede a comunicarglielo.

La differenza principale tra un cluster di tipo Beowulf ed i cluster di Workstation (COW) risiede nel fatto che il Beowulf si comporta più come una singola entità che come un gruppo di workstation aggregate. Nella maggior parte dei casi i client sono sprovvisti di tastiere e monitor e per accedervi si devono utilizzare strumenti quali il login remoto o terminali seriali. Un nodo Beowulf, pertanto, può essere pensato come un pacchetto composto da CPU e memoria che può essere inserito nel cluster, proprio come una CPU oppure un modulo di memoria può esserlo all'interno della piastra madre¹¹ di un personal computer.

Da questa riflessione si evince che Beowulf non è un pacchetto software, né un tipo di rete, né una particolare implementazione del kernel di GNU/Linux, ma una tecnica di clustering per realizzare supercomputer paralleli virtuali. Fermo restando che esistono delle modifiche software al kernel di GNU/Linux, delle utilità di sistema e delle librerie tipo PVM ed MPI che rendono l'architettura Beowulf più performante, più facile da configurare e molto più pratica e flessibile da utilizzare, possiamo considerare come Beowulf anche una semplice coppia di macchine GNU/Linux che utilizzano una distribuzione standard, condividono una porzione di filesystem e che siano in grado di eseguire comandi utilizzando shell remote (rsh). In questo caso avremo un sistema Beowulf composto da due soli nodi.

¹¹ Piastra madre: è la piastra principale di un personal computer, sulla quale vengono inserite la/e CPU, la memoria RAM ed eventuali schede di espansione.

1.8 Classificazione dei Cluster Beowulf

I sistemi Beowulf possono essere realizzati utilizzando una grande varietà di parti comuni ed in alcuni casi vengono utilizzati componenti non comuni (cioè prodotti da una singola ditta) per incrementare le prestazioni del sistema.

Per schematizzare i differenti sistemi e rendere più chiara la trattazione possiamo dividere i sistemi Beowulf in due classi:

- Beowulf di Classe 1: sono macchine costituite da componenti comuni e di facile reperibilità, ovvero i componenti possono essere reperiti attraverso normali negozi d'informatica oppure da cataloghi pubblicitari a carattere nazionale o internazionale.

I vantaggi dei sistemi di classe 1 sono che l'hardware è reperibile da molte fonti, il che implica prezzi bassi e manutenzione semplificata, che non vi è dipendenza da un singolo rivenditore di hardware e che il supporto a livello driver¹² è basato normalmente su standard di mercato (SCSI, Ethernet ecc).

Lo svantaggio di un sistema di Classe 1 è che la richiesta di maggiori prestazioni potrebbe necessitare di hardware di Classe 2, vanificando la convenienza dell'investimento iniziale.

- Beowulf di Classe 2: rientrano in questa categoria tutti i sistemi che fanno uso di componenti non di uso comune, sia in quantitativi modici che in quantitativi massicci. I vantaggi di tale soluzione sono tutti legati alle performance, mentre gli svantaggi risiedono nella qualità dei driver e nella dipendenza da un singolo rivenditore hardware, oltre al fatto che sono molto più costosi di un cluster di Classe 1.

¹² Si definisce come driver un particolare programma del sistema operativo necessario per interfacciare il sistema con l'hardware specifico. Per un dettaglio dei driver utilizzati all'interno delle configurazioni del cluster si vedano i listati dei file “.config” presenti in appendice.

Una classe non è necessariamente migliore dell'altra, tutto dipende dal tipo di applicazione e dal budget.

1.9 HA come estensione di un cluster HPC

L'introduzione di sistemi di supercalcolo in contesti commerciali o comunque in ambiti differenti dal tipico ambiente di ricerca sta determinando una convergenza sempre più forte tra l'universo HPC e quello HA. Poiché la soluzione ibrida HPC + minicomputer nel contesto commerciale, risulta inapplicabile a causa di una richiesta monetaria troppo esosa ed a causa di una disponibilità di spazi sempre più limitata, si rende indispensabile rivolgere l'attenzione a soluzioni alternative. Per questo motivo in moltissimi contesti i server HA stanno diventando il complemento naturale dei cluster HPC e stanno innescando un nuovo interesse nei confronti di una tecnologia valida, ma per troppi anni relegata a nicchie di mercato.

In configurazioni semplici si cerca di completare la struttura di un cluster HPC con fileserver ridondanti, gateway multipli, sistemi di autenticazione ridondanti, ecc.

In soluzioni più complesse si punta ad integrare nei gateway ad alta disponibilità soluzioni di virtualizzazione dei servizi e di bilanciamento del carico. L'adozione di tali tecnologie, abbinata a software in grado di monitorare il carico dei server del cluster, permette un utilizzo ottimale delle risorse dell'intero sistema.

Ad esempio, se nel caso di cluster composti da centinaia di nodi ed utilizzati da decine di persone contemporaneamente non si adottassero strategie di allocazione/redirezione dinamica delle richieste di elaborazione ci si potrebbe trovare di fronte ad un sottoutilizzo della potenza di calcolo conseguente ad un semplice sovraccarico delle workstation adibite all'invio delle richieste di elaborazione al sistema.

All'interno del presente lavoro di tesi si analizza la creazione di una soluzione HPC dotata di funzionalità HA.

La struttura è composta da due sezioni:

- la prima riguarda l'alta disponibilità ed è composta da due nodi basati su tecnologia Intel Pentium 3 operante a 800 Mhz, 512MB di memoria SDRAM e da tre schede di rete per gestire rispettivamente il traffico in ingresso, l'interscambio delle informazioni di stato tra i nodi HA, ed il traffico verso i nodi del cluster.
- La seconda riguarda l'HPC ed è composta da tre nodi basati su doppio processore AMD Athlon MP 1800+ coadiuvato da 1 Gbyte di ram DDR e da due schede di rete. La prima operante a 100 Mbit/s per permettere l'avvio del nodo in modalità diskless e la seconda operante a 1 Gbit/s per l'interscambio delle informazioni da elaborare.

La sezione HA è stata realizzata per fornire servizi di file server, autenticazione, bilanciamento di carico, ed inizializzazione dei nodi diskless, mentre la sezione HPC è stata realizzata per l'elaborazione massiva dei dati provenienti dal mammografo.

CAPITOLO 2

HPC A 32 BIT

In questo capitolo si analizza l'installazione e la configurazione di uno dei nodi a 32 bit che compongono il cluster. Come già specificato, il sistema operativo sul quale verrà realizzata l'intera soluzione è GNU/Linux e su di esso verranno installati i vari pacchetti software che compongono sia la porzione HPC che la porzione HA del cluster.

2.1 Progettazione

La progettazione di un cluster HPC risulta estremamente delicata, dal momento che ogni decisione in fase di progettazione può avere pesanti ripercussioni sulle performance e sulla “mantenibilità” dell'intero cluster.

Per ottenere un sistema flessibile ed al tempo stesso performante, si è scelto di utilizzare il sistema operativo GNU/Linux che è noto per la sua flessibilità, per i suoi bassi costi di gestione e per essere liberamente scaricabile ed adattabile alle esigenze dell'utente. In particolare, si è scelto di utilizzare la distribuzione¹³ Linux denominata Gentoo¹⁴, che ha il vantaggio di essere estremamente ottimizzabile in funzione della tipologia di hardware adottato.

Al fine di semplificare la progettazione dell'intero sistema distribuito, si è scelto di suddividerla in due sezioni distinte:

- progettazione del server HPC;
- progettazione dei client o nodi HPC;

¹³ Distribuzione: Nel mondo di GNU/Linux l'abbinamento del kernel creato originariamente da Linus Torvalds e delle utility di sistema genera quello che in gergo viene chiamata “distribuzione linux”.

¹⁴ Gentoo: www.gentoo.org / www.gentoo.it

Per quanto concerne la progettazione del server HPC, si è scelto di realizzare un server in grado di gestire oltre ai normali servizi di un nodo server Beowulf, anche i servizi necessari al funzionamento di nodi client di tipo diskless.

Nella fattispecie si sono attivati servizi quali:

- NFS per la gestione del file-system condiviso;
- DHCP per l'assegnazione automatica degli indirizzi IP;
- TFTP per il boot.

Per quanto concerne la progettazione dei nodi client, si è provveduto all'installazione ed alla configurazione del sistema su un nodo client dotato di disco e successivamente si è provveduto a trasferire l'immagine sul nodo server per essere adattata ad una struttura completamente diskless. Tale scelta è stata dettata dall'esigenza di abbassare il livello di complessità dell'intera installazione, dal momento che presenta diversi passaggi critici.

Come risultato si è ottenuto un nodo master in grado di agire sia da frontend del cluster Beowulf, sia da server di rete in grado di:

- assegnare indirizzi IP;
- fornire un'immagine di boot;
- autoconfigurare l'ambiente;
- fornire un file-system di root (/) a tutti i nodi diskless del cluster.

Pertanto i nodi al momento dell'avvio eseguono le seguenti operazioni:

- richiedono un indirizzo IP al server;
- caricano un'immagine di boot;
- richiedono i dati per procedere all'autoconfigurazione dell'ambiente;

- montano il file-system di root (/) fornito dal server;
- caricano il sistema operativo ed i servizi.

2.2 Hardware adottato

Il sistema è stato realizzato utilizzando due nodi a basse performance e tre nodi ad alte performance per la sezione HPC.

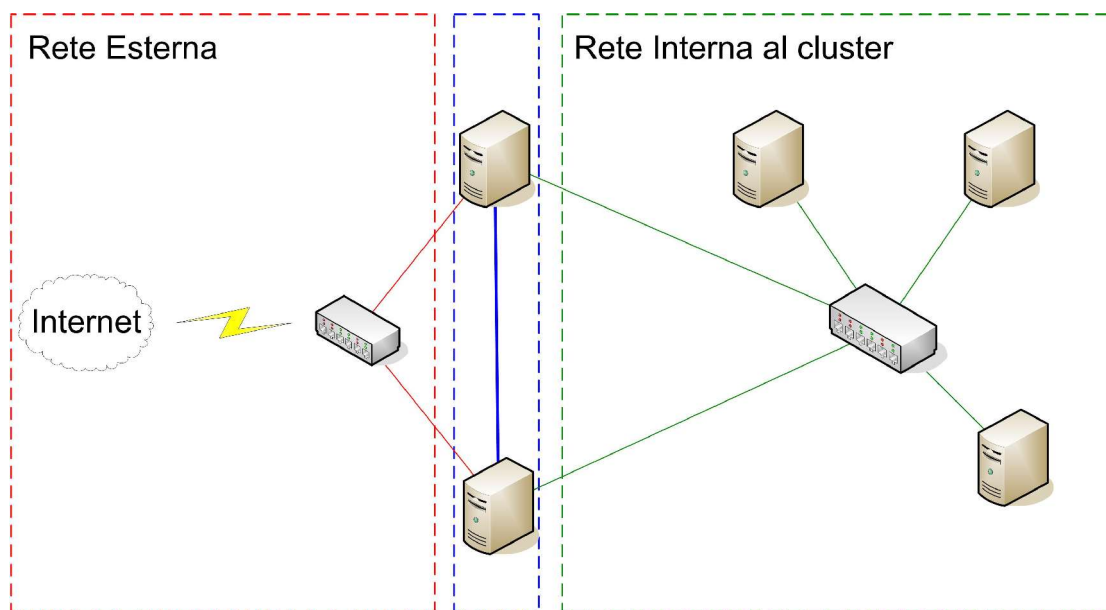


Figura 2.1: Struttura del cluster

L'hardware dei due server in configurazione HA, basato su tecnologia Intel Pentium 3, allo stato attuale, è da considerarsi obsoleto, tuttavia, coadiuvato dall'ottimo sistema operativo GNU/Linux si è dimostrato ancora valido ed in grado di esprimere un livello di performance accettabile. La piastra principale, basata su tecnologia Intel BX, è dotata di CPU Pentium 3 ed opera a 800MHz con 256kb di cache e FSB¹⁵ di 133MHz. La

¹⁵ FSB: (Front Side Bus): canale di comunicazione fra il chipset della scheda madre e il processore. Valori più alti di frequenza di funzionamento riducono il collo di bottiglia che si può presentare nella comunicazione tra il processore ed il resto del sistema.

memoria installata nel sistema è 256 Mbyte ed è basata sulla tecnologia SDRAM¹⁶ senza controllo e correzione di errore. Il sottosistema dischi è basato su un hard disk IBM da 60 Gbyte funzionante a 7200 giri/min. Completano il sistema 3 schede di rete utilizzate per l'interconnessione con l'esterno, per la sincronizzazione dei dati ed i valori di stato tra i nodi ad alta disponibilità e per la connessione con i nodi del cluster HPC.

L'hardware dei nodi HPC è basato su piattaforma AMD Athlon MP che permette di avere due processori operanti in configurazione simmetrica (SMP¹⁷). Nel nostro caso le macchine erano dotate di CPU AMD Athlon MP 1800+ dotate di 256kb di cache, un front side bus di 266MHz ed 1GB di memoria DDR¹⁸ senza correzione di errore. Completano il sistema una coppia di schede di rete a 100 ed a 1000 MBit/s, utilizzate per il caricamento dei dati e del sistema dal file server. Come in tutte le unità di questo tipo, la sezione grafica non è presente dato che il nodo dovrà esprimersi unicamente nel calcolo puro e non dovrà essere utilizzato come stazione di lavoro degli utenti.

2.3 Scelta del sistema operativo

Il componente principale dell'intero sistema distribuito è ovviamente il sistema operativo. Per la scelta di questo componente chiave sono state valutate diverse alternative sia di tipo closed source, come Microsoft Windows, sia di tipo open source, quali GNU/Linux, nelle sue diverse distribuzioni, sia le diverse varianti di BSD.

Le soluzioni di tipo closed sono state immediatamente escluse, sia per motivi di tipo economico, dato che i software di tipo closed source sono generalmente a pagamento, sia per problematiche legate alla flessibilità dell'installazione, dato che, non essendo

¹⁶SDRAM: acronimo indicante la Synchronous Dynamic RAM

¹⁷SMP: (Simultaneous Multi Processing) consente l'esecuzione contemporanea di istruzioni nei diversi processori disponibili.

¹⁸DDR Ram: acronimo indicante Double Data Rate-Synchronous Dynamic RAM

disponibile il codice sorgente delle singole componenti del sistema, risulta pressoché impossibile compiere modifiche di rilievo al sistema.

Successivamente, sono state analizzate le diverse caratteristiche dei sistemi BSD, prime fra tutte la sicurezza e l'essenzialità che stanno alla base di tali sistemi, tuttavia, si è rilevato un livello di supporto molto limitato se paragonato a GNU/Linux. Si è dunque deciso di optare per il sistema operativo GNU/Linux. In particolar modo, si è deciso di utilizzare la metadistribuzione¹⁹ Gentoo Linux che permette di avere un sistema sempre aggiornato ed ottimizzato al massimo in funzione dell'hardware disponibile. Rispetto alle altre, questa distribuzione ha la caratteristica di essere fornita in linea preferenziale in formato sorgente. Tuttavia, questa modalità operativa presenta un difetto: la compilazione, specie se di tutto il sistema, richiede molto tempo. Al contempo presenta il notevole vantaggio di poter ottenere un sistema operativo ottimizzato in base all'hardware adottato ed alle esigenze dell'utente. Come si può facilmente intuire, il sistema così generato è specificatamente creato per la macchina su cui gira, ed è quindi ottimizzato in termini di prestazioni e componenti installati.

Al contrario di quello che si sarebbe portati a pensare, Gentoo non è una distribuzione creata unicamente per gli utenti avanzati, dal momento che è tra le più ricche di documentazione ed in quanto operazioni d'installazione e disinstallazione spesso consistono solamente nel digitare un comando da shell (es: per l'installazione di un pacchetto si utilizza *emerge nome-pacchetto*, dove *nome-pacchetto* è il nome del software che si intende installare).

¹⁹ Metadistribuzione: distribuzione basata sui sorgenti che può, attraverso il settaggio di alcune opzioni, generare tipologie di sistemi operativo dalle caratteristiche diametralmente opposte. Si pensi ad esempio ad una distribuzione per computer embedded, ad una per i server, ad una per i computer desktop.

2.4 Installazione del sistema base

Il processo d'installazione dell'intero sistema è iniziato il 19 Marzo 2004 utilizzando l'ultima release disponibile attraverso i mirror²⁰ di Gentoo linux, ovvero la 2004.0 . Tuttavia, durante lo sviluppo dell'intero sistema si sono fatti numerosi aggiornamenti, mantenendo di fatto un sistema perfettamente in linea con gli ultimi sviluppi.

Analogamente a quanto segnalato per l'installazione dei singoli pacchetti, aggiornare una distribuzione come Gentoo Linux non è nulla di più complicato che lanciare un semplicissimo comando da shell quale è *emerge -uD world*²¹.

Il processo d'installazione può essere suddiviso in diverse fasi operative che rappresentano le tappe essenziali per ottenere un sistema perfettamente funzionante:

- Fase 1: avvio e creazione del file-system;
- Fase 2: Compilazione del sistema base;
- Fase 3: installazione del sistema base e utility di sistema.

Fase 1: avvio e creazione del file-system

L'installazione delle distribuzioni GNU/Linux può avvenire in molteplici modi:

- via rete (locale o internet);
- utilizzando supporti di memorizzazione quali floppy o CD;
- clonando installazioni esistenti.

²⁰ Un mirror è una copia fedele di un sito internet. I mirror, spesso, vengono utilizzati per i siti con livelli di traffico molto elevati al fine di non sovraccaricare i server.

²¹ Prima di lanciare tale comando è buona norma aggiornare il database locale dei pacchetti della distribuzione con *emerge sync*. Tuttavia tale operazione può necessitare di parecchio tempo, in funzione del quantitativo di tempo trascorso dopo l'ultimo aggiornamento.

Il metodo classico e più diffuso rimane comunque l'installazione da CD. Nel caso di Gentoo Linux, data la sua atipicità derivante dalla compilazione del sistema direttamente sulla macchina target, sono necessarie tre fasi successive prima di avere un sistema perfettamente funzionante. Tali fasi, che prendono il nome di stage, servono a compilare ed installare porzioni differenti del sistema.

All'interno dello stage1 si esegue la compilazione del compilatore C e delle relative librerie, nello stage2 si compilano le porzioni chiave del sistema operativo, fino ad arrivare allo stage3 in cui è già tutto compilato tranne il boot loader e le applicazioni non strettamente necessarie all'utilizzo base del sistema operativo.

Per velocizzare il processo d'installazione, gli sviluppatori di Gentoo Linux hanno inserito all'interno dei CD d'installazione degli archivi contenenti gli stage base del sistema, permettendo di partire anche da livelli successivi al primo.

Installando Gentoo Linux quindi ci si trova nella condizione di dover scegliere anche quale stage utilizzare. Tale scelta, generalmente, viene effettuata sia considerando le potenzialità della macchina ed il conseguente lasso di tempo necessario a terminare l'operazione, sia considerando che ogni stage contiene i precedenti compilati con opzioni generiche. Pertanto, se decidiamo di partire dallo stage2, alla fine dell'elaborazione avremo tutti i pacchetti dello stage1 compilati con opzioni generiche ed i pacchetti propri dello stage2 compilati con opzioni specifiche, tuttavia, avremo risparmiato circa quattro ore.

A solo scopo esemplificativo, si consideri che la compilazione delle “glibc”, su una macchina dual athlon MP 1800+ con 1 Gb di Ram, richiede mediamente oltre due ore di elaborazione continua.

Nell'installazione del sistema Gentoo Linux, su ogni tipologia di hardware (Intel Pentium3 ed AMD Athlon MP), si è scelto di partire dallo stage1 e ricompilare per intero il sistema, cercando di ottimizzare al massimo le performance di ogni porzione.

Il primo passo è stato quello di scaricare dai mirror di <http://www.gentoo.org> l'immagine ISO²² del liveCD²³ contenente un sistema minimale capace di far partire la macchina e gli archivi precompilati degli stage1, 2 e 3.

Successivamente, si sono masterizzate due copie identiche del liveCD e si è fatto partire il sistema minimale contemporaneamente su uno dei nodi master HPC (Pentium 3) e su un nodo slave HPC (Dual Athlon MP) temporaneamente dotato di hard disk.

Prima del "bootstrap" del sistema operativo si è provveduto a passare al kernel alcune opzioni per meglio adattarlo alle caratteristiche dell'hardware. In particolare, data la scarsità delle risorse grafiche di entrambe le macchine, si è provveduto a disabilitare il boot della console grafica (basata su framebuffer) e nel caso del nodo slave HPC si è provveduto ad abilitare il supporto multiprocessore attraverso il parametro smp.

Una volta completata la procedura di boot, si è provveduto a selezionare la mappatura corrispondente alla tastiera italiana attraverso il comando *loadkeys it* e si è inizializzata la rete in modo tale da poter eseguire il download dei diversi pacchetti richiesti dal sistema di gestione dei pacchetti, denominato "portage"²⁴.

Successivamente, si è approntato uno schema di partizionamento, preparando, di fatto, il sistema all'installazione vera e propria.

²²ISO: è l'abbreviazione di International Organization for Standardization. Normalmente in ambito GNU/Linux con la parola ISO si indica in realtà lo standard ISO9660 tipo dei supporti CD-ROM.

²³ LiveCD: è un termine utilizzato per riferirsi ad un CD ROM contenente il sistema operativo GNU/Linux utilizzabile direttamente da CD senza la necessità di dover installare nulla su memorie di massa locali.

²⁴ Portage è il database delle applicazioni installabili direttamente utilizzando il comando emerge.

Tale database comprende, allo stato attuale, svariate migliaia di applicazioni installabili per un totale di oltre 30Gb di dati, numero in rapida crescita dato il livello di diffusione che sta raggiungendo Gentoo Linux.

Per eseguire il partizionamento, si è usato il tool di sistema fdisk²⁵ attraverso alcuni semplici menù per creare, modificare ed eliminare ogni partizione fisica o logica presente all'interno del sistema.

Lo schema di partizionamento è quello specificato nella seguente tabella:

Nome partizione	Dimensione (Mb)	Mount point	Tipo	Opzioni
hda1	100	/boot	ext2 ²⁶	noauto
hda2	1000		swap	
hda3	9000	/	ext3 ²⁷	noatime
hda4	30000	/diskless	ext3	noatime

Tabella 2.1: Schema di partizionamento

Dopo aver creato le partizioni secondo lo schema indicato, si è provveduto a creare i file-system su ogni partizione, in particolare:

`mkswap /dev/hda2` per la partizione di swap

`mkfs.ext3 /dev/hda1` per la partizione /boot

`mkfs.ext3 /dev/hda3` per la partizione /

si è tralasciata la creazione del file-system sulla partizione hda4, che dovrà essere inizializzata in un modo particolare per permettere la creazione di un file-system replicato via rete (che verrà poi utilizzato all'interno della porzione HA del sistema).

²⁵ Fdisk: è lo storico tool per il partizionamento degli hard disk.

²⁶ Ext2: file-system storico usato sulla gran parte delle installazioni Linux prima della creazione di ext3 che ne rappresenta la diretta evoluzione con la sola aggiunta del journaling.

²⁷ Ext3: File-system journaled diretta evoluzione di ext2. La caratteristica journaled fornisce la capacità di evitare inconsistenze nei dati in caso di spegnimento accidentale o crash di sistema. Pertanto, in caso di spegnimento nel regolare, al primo "montaggio" del file-system il sistema è in grado di ripristinare lo stato consistente dei dati annullando le operazioni non completate prima del blocco.

Fatto questo, si è provveduto ad attivare la partizione di swap col comando *swapon /dev/hda2* e a montare le diverse partizioni sui rispettivi mount point con il comando *mount /dev/hdax /mnt/gentoo/path*, dove */path* corrisponde al percorso relativo a ogni partizione come specificato nella tabella 1.

Fase 2: Compilazione del sistema base

Al fine di preparare il sistema alla compilazione del sistema base, bisogna provvedere a scompattare il file contenente lo stage desiderato nell'apposita directory radice, dove precedentemente abbiamo montato la partizione di root.

Pertanto si è provveduto a:

- entrare nella partizione di root attraverso il comando `cd /mnt/gentoo`;
- scompattare lo stage1 presente nel cd di installazione:
`tar xvjf /mnt/cdrom/stages/stage1*`
- scaricare l'elenco dei mirror dei sorgenti dei pacchetti:
`mirrorselect -a -s4 -o >> /mnt/gentoo/etc/make.conf`
- montare il file-system speciale `/proc`²⁸:
`mount -t proc none /mnt/gentoo/proc`
- creare un file `/mnt/gentoo/etc/resolv.conf` contenente gli indirizzi dei server DNS da utilizzare durante gli scaricamenti da internet;
- montare il file-system speciale `/dev`:
`mount -t devfs none /mnt/gentoo/proc`
- entrare nell'ambiente chroot²⁹ dello stage1:
`chroot /mnt/gentoo`

²⁸ Il file-system `proc` è un utilizzato da GNU/Linux per permettere di visionare tutte le informazioni riguardanti lo stato attuale del funzionamento del sistema operativo. E' costituito da un insieme di file e directory ed essenzialmente ricalca la filosofia di GNU/Linux nella quale ogni cosa è un file.

²⁹ Chroot: l'ambiente `chroot` è un sottoambiente dell'ambiente principale della distribuzione GNU/Linux dalla quale è isolato in modo perfetto (equivalentemente ad un sistema nel sistema).

A questo punto la directory radice del sistema è diventata `/mnt/gentoo` per cui tutte le modifiche apportate sono relative ai file sotto tale percorso e quindi anche a quello che sarà il sistema operativo finale della macchina.

Pertanto, ora è possibile aggiornare l'ambiente di gentoo con il comando `env-update` seguito da `source /etc/profile` e si può procedere allo scaricamento del database dei pacchetti gentoo, comunemente definito portage, attraverso il comando:

```
emerge sync
```

Tale operazione deve essere compiuta ogni qual volta sia necessario aggiornare l'intero sistema oppure sia necessario utilizzare dei pacchetti aggiunti solo recentemente.

Successivamente, si è provveduto a creare due distinti file di configurazione per il sottosistema di compilazione:

- il primo sul nodo server HPC, ottimizzato per macchine a singolo processore basate su architettura Intel Pentium 3;
- il secondo, ottimizzato per architetture AMD dual Athlon MP.

Tale file, denominato `/etc/make.conf`, risulta essere di vitale importanza in una distribuzione come Gentoo Linux. Infatti, all'interno di esso vengono inserite tutte le variabili che determinano come il sistema debba essere compilato e dove debbano essere reperite le informazioni per la compilazione stessa.

Le differenze salienti tra i due file risiedono essenzialmente nelle opzioni di compilazione dell'architettura e nel numero di “job” gestibili contemporaneamente in fase di compilazione (la guida consiglia di selezionare un numero di job pari al numero di CPUs+1).

Per quanto riguarda l'architettura Intel Pentium 3 sono stati utilizzati i seguenti parametri:

```
CHOST="i686-pc-linux-gnu"
CFLAGS="-march=pentium3 -O2 -pipe -fomit-frame-pointer"
CXXFLAGS="-march=pentium3 -O2 -pipe -fomit-frame-
pointer"
```

Mentre per l'architettura AMD dual Athlon MP sono stati utilizzati i seguenti parametri:

```
CHOST="i686-pc-linux-gnu"
CFLAGS="-march=athlon-mp -O3 -pipe -fomit-frame-
pointer"
CXXFLAGS="-march=athlon-mp -O3 -pipe -fomit-frame-
pointer"
```

Successivamente, si è lanciata la compilazione del sistema operativo attraverso i comandi:

```
sh /usr/portage/scripts/bootstrap.sh
```

ed

```
emerge system.
```

Fase 3: installazione del sistema base e utility di sistema

Allo stato attuale dell'installazione, pur avendo atteso un lungo periodo a causa della compilazione, il sistema è del tutto privo di due componenti fondamentali: il *kernel* ed il *boot loader*.

La traduzione letterale del termine “kernel” è “nucleo” ed in effetti il kernel è da considerarsi come il centro del sistema operativo, dato che è a diretto contatto con l'hardware, nei confronti del quale agisce da interprete delle richieste, e con il software, del quale gestisce ogni aspetto, dall'allocazione della memoria, all'interazione con l'hardware ecc.

Quando si vogliono spiegare le funzionalità di un sistema operativo complesso, quale è Linux, spesso si utilizza la “metafora della cipolla”. Come è noto la cipolla è costituita da diversi strati sovrapposti: al centro troviamo l'hardware, immediatamente sopra c'è il kernel, poi il software di sistema ed ancora esternamente troviamo i programmi applicativi utilizzati dagli utenti.

Come si può facilmente dedurre, ogni strato rappresenta un livello di astrazione che fa da interprete fra lo strato più esterno e quello più interno ad esso e rende possibile l'astrazione funzionale che consente all'utente di usare la macchina, senza dover comunicare utilizzando linguaggi innaturali.

A questo punto non rimane altro che scrivere il file di configurazione */etc/fstab*³⁰, contenente i riferimenti alle partizioni di sistema, e configurare l'orologio di sistema, adattandolo all'area geografica italiana (+1 rispetto a Greenwich) attraverso il comando:

```
In -s /usr/share/zoneinfo/Europe/Rome /etc/localtime
```

³⁰ Fstab: nei sistemi GNU/Linux permette di tenere traccia delle partizioni di sistema e permette inoltre di attivare le stesse con un determinato set di parametri.

Ora il sistema risulta pronto per procedere con la compilazione del kernel. Pertanto, non rimane altro che scegliere tra le versioni messe a disposizione dagli sviluppatori di Gentoo Linux.

Per il sistema in oggetto, si è scelto di utilizzare il kernel “vanilla”, ovvero la versione ufficiale distribuita attraverso il sito <http://www.kernel.org>, pertanto si è lanciato il comando:

```
emerge vanilla-sources
```

e si sono installati i sorgenti utilizzando l'ebuild³¹ ufficiale di Gentoo. Al termine dell'installazione, si è provveduto ad installare l'utility *genkernel* che permette di automatizzare le fasi di compilazione ed installazione dell'immagine del kernel, mettendoci al riparo da eventuali dimenticanze ed errori. L'installazione di tale utility è stata effettuata, come di consueto, con il comando:

```
emerge genkernel
```

Successivamente, si è provveduto a lanciare il comando *genkernel --config* e si è effettuata la configurazione delle varie componenti del kernel specificando, ad esempio, il tipo di processore utilizzato (Intel Pentium 3 per il nodo master HPC ed AMD Athlon MP per il nodo slave HPC), il limite massimo di memoria allocabile (che per il nodo slave HPC è pari ad 1GB), la presenza o meno del supporto SMP (disattivato sul nodo master HPC, attivato sul nodo slave HPC), abilitando il supporto per l'assegnazione degli indirizzi di rete attraverso protocollo dhcp ed il supporto di file-system di root montati via rete ed eliminando successivamente tutti i driver relativi a periferiche non presenti nel sistema.

³¹ Ebuild: un ebuild è uno script bash creato per il sistema di gestione a pacchetti, denominato portage, della distribuzione Gentoo Linux. Ogni applicazione nel portage ha uno script scritto per essa.

Al termine della fase di configurazione, si è utilizzata la medesima utility per generare l'immagine del kernel e per installare tutti i moduli nelle posizioni corrette.

Al termine dell'operazione, si è ottenuto un sistema a tutti gli effetti funzionante, dato che il kernel ed il set di utility installate sono in grado di gestire la macchina senza l'ausilio di ulteriori programmi.

Tuttavia, per rendere più agevole le fasi di boot, si è scelto d'installare anche un boot loader residente sul disco, nella fattispecie grub, ed un set minimale di applicazioni per gestire le operazioni più comuni:

- un demone per l'esecuzione programmata di programmi: cron;
- un demone per gestire i flussi di log provenienti dal kernel: syslog-ng;
- un'utility per l'autodetect dell'hardware: hotplug;
- le utility per il check periodico dei file-system utilizzati nel sistema: e2fsprogs.

Inoltre:

- si sono resi avviabili i servizi precedentemente elencati utilizzando il comando `rc-update add <nomeservizio> default` (che indica al sistema di far partire il tal servizio al momento del boot del sistema);
- si è configurata la mappa della tastiera, attraverso la modifica della variabile `keymap` all'interno del file `/etc/rc.conf`;
- si è configurato l'hostname della macchina con il comando:
`echo "nomemacchina" > /etc/hostname`
- si è configurato il supporto base della rete per permettere un accesso veloce ai sorgenti dei pacchetti gentoo in attesa di approntare la configurazione definitiva dell'infrastruttura di rete;

- Si è configurata la password di root con il comando:

```
passwd
```

Per finire, dopo aver terminato la sessione *chroot* attivata all'inizio del processo d'installazione (con il comando *exit*), si è provveduto a smontare le partizioni *proc* e *dev* con il comando *umount /mnt/gentoo/proc* e *umount /mnt/gentoo/dev* e tutte le altre partizioni montate con *umount -a* e finalmente si sono riavviate le due macchine appena installate (il nodo server HPC ed il nodo client HPC che sarà il prototipo utilizzato per la creazione dell'immagine diskless dalla quale partiranno tutti i nodi slave HPC).

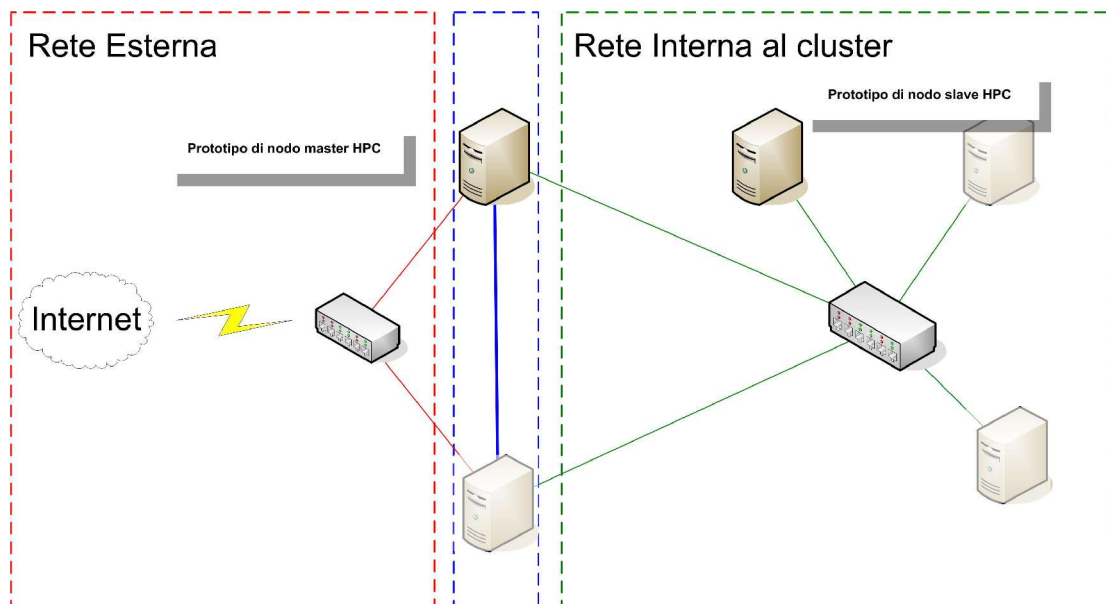


Figura 2.2: Prototipi di nodo master HPC e nodo slave HPC

2.5 Creazione dell'infrastruttura di rete

L'infrastruttura di rete è stata realizzata partizionando l'intera struttura in tre zone distinte:

- la rete esterna (ovvero la LAN del laboratorio di Biofisica delle Reti Neurali);
- la rete interna al cluster;
- il link heartbeat (che servirà per far comunicare i due server HPC ed i servizi ad alta disponibilità quando saranno installati sulle suddette macchine);

La rete interna al cluster è stata a sua volta divisa in due porzioni:

- la prima a 100Mbit/s realizzata utilizzando uno switch 10/100Mbit/s a 24 porte della 3COM per eseguire il boot via rete dei nodi diskless e per l'interscambio dei files di sistema;
- la seconda ad 1 Gbit/s realizzata utilizzando uno switch da 1Gbit/s a 8 porte prodotto da Corega per lo scambio dei dati utilizzati durante le elaborazioni e come bus MP.

Gli indirizzi di rete utilizzati e la struttura di rete sono riassunti nella tabella e nella figura che seguono.

<i>IP</i>	<i>Nome Host</i>
10.0.3.1	master1
10.0.3.2	master2
10.0.3.11	slave1
10.0.3.12	slave2
10.0.3.13	slave3

Tabella 2.2: Indirizzi IP interni al cluster

2.6 Configurazione dei servizi DHCP, PXE e NFS per i nodi diskless

Come precedentemente accennato, si è deciso per questioni di praticità di amministrazione di non dotare i client HPC di hard disk, ma di caricare i file necessari al loro funzionamento attraverso la connessione di rete a 100Mbit/s.

Pertanto, si è reso necessario attivare, sul nodo master HPC, un insieme di servizi in grado di:

- assegnare un indirizzo IP;
- inviare un kernel valido in grado di gestire il file-system di root via rete;
- inviare i parametri per l'autoconfigurazione dei client;
- “montare” un file-system di root condiviso in rete.

Un architettura di tipo x86 normalmente viene utilizzata per un utilizzo “personale”, pertanto, spesso non è dotata di particolari accortezze come il supporto per il boot via rete oppure via device che non siano riconducibili alle classiche unità di memorizzazione di massa.

Al fine di trasformare un comune pc in un vero e proprio nodo diskless è necessario dotarlo di un sistema in grado di caricare sia i dati di configurazione, sia i componenti chiave del sistema operativo, utilizzando unicamente la rete locale.

Tali sistemi normalmente sono di due tipi:

- boot mediante eprom dotate di software etherboot;
- boot mediante sottosistema PXE.

La prima tipologia prevede l'utilizzo di costose e delicate eprom da installare su apposite schede di rete dotate di zoccolo per eprom, mentre la seconda tipologia prevede

l'utilizzo di un microcodice inserito nell'hardware della scheda, pertanto non aggiornabile, e di una controparte liberamente modificabile ed utilizzabile da inviare al client utilizzando il protocollo TFTP nelle prime fasi dell'avvio del sistema.

Dato che le nostre schede a 100Mbit/s supportano il PXE, si è deciso di optare per la seconda soluzione evitando così di dover reperire costose eprom.

Il boot di client diskless utilizzando il sistema PXE può essere riassunto come segue:

<i>Lato Server HPC</i>	<i>Lato Client HPC diskless</i>
	richiesta dell'indirizzo IP
invio dell'indirizzo IP	
	<ul style="list-style-type: none"> • inizializzazione dell'interfaccia di rete con l'indirizzo IP inviato; • richiesta del codice PXE via TFTP.
invio del codice PXE via TFTP	
	<ul style="list-style-type: none"> • esecuzione del codice PXE ed inizializzazione del sottosistema di rete esteso; • richiesta dell'immagine del kernel da caricare.
invio dell'immagine del kernel via TFTP	
	<ul style="list-style-type: none"> • caricamento dell'immagine del kernel; • richiesta dell'immagine initrd corrispondente al kernel caricato.
invio dell'immagine initrd via TFTP	
	mount del file-system di root utilizzando il protocollo NFS

Tabella 2.3: Boot di client diskless col sistema PXE

Una volta completata la configurazione del boot via rete che può essere visionata nell'appendice, si è provveduto ad affinare l'intero processo ricercando una soluzione che rendesse il più possibile flessibile e scalabile l'intera infrastruttura del cluster e che riducesse a zero la ridondanza dei file di sistema dei nodi client HPC.

Tipicamente, infatti, ogni nodo carica dal server una porzione generica ed una porzione specifica. Nella porzione generica sono contenuti tutti i binari³² (es: /sbin /bin /usr etc), nella porzione specifica sono contenuti i file di configurazione del singolo nodo. Pertanto, in sistemi composti da decine di nodi, spesso ci si trova ad avere una proliferazione di directory quali /etc-slave1, /etc-slave2 creando ridondanza e rendendo il sistema sempre più “immantenibile”.

Per eliminare questa problematica di amministrazione, si è realizzato uno strato di virtualizzazione in grado di autoconfigurare i nodi slave HPC e generare dinamicamente al boot tutti i parametri di configurazione considerabili come specifici del singolo nodo.

2.7 Realizzazione del prototipo della Single System Image e realizzazione di un middle-layer per l'autoconfigurazione dei nodi slave HPC

Per comprendere al meglio l'importanza di un componente quale lo strato di virtualizzazione di un ambiente di boot per nodi diskless dobbiamo ripercorrere velocemente le fasi realizzative descritte fino a questo momento:

- per prima cosa è stato realizzato il prototipo di un nodo server HPC e di un nodo client HPC;

³² Con il termine “binario” si intende la versione compilata in linguaggio macchina e quindi direttamente eseguibile da un calcolatore.

- in secondo luogo sono stati copiati i dati presenti sull'hard disk del prototipo del nodo client HPC in una apposita directory sul nodo server HPC (come si detto si vuole arrivare alla realizzazione di nodi client HPC completamente privi di unità di memorizzazione di massa);
- successivamente è stato predisposto un ambiente in grado di permettere il boot via rete dei nodi diskless (a tal fine sono stati attivati i servizi DHCP e TFTP sul nodo server HPC).

A questo punto ci troviamo con un nodo server HPC in grado di gestire dei nodi client HPC, ma ci troviamo a dover replicare n volte (n = al numero di nodi client HPC) le directory contenenti i parametri di configurazione dei client HPC. Tale proliferazione, oltre ad aumentare il livello di complessità dell'amministrazione, può sicuramente portare ad errori di configurazione.

Dopo un'attenta analisi del processo di avvio dei nodi client HPC, si è individuato un adeguato punto d'intervento atto ad eseguire l'integrazione di uno strato virtuale con la classica struttura delle distribuzioni GNU/Linux.

Nella fattispecie, si è deciso di integrare il tutto all'interno dell'immagine initrd inviata subito dopo il caricamento del kernel del sistema. Tale immagine, che normalmente serve da complemento all'immagine del kernel per il caricamento dinamico dei driver necessari al boot del sistema, è stata completamente riscritta fino a diventare una vera e propria mini-distribuzione linux in grado di compiere ogni tipo di operazione.

Per limitare la dimensione dell'immagine finale, si è deciso di compilare tutti i programmi necessari allo strato di virtualizzazione, utilizzando le librerie C denominate uclibc. Tali librerie a differenza delle classiche glibc, comunemente utilizzate in tutte le distribuzioni GNU/Linux, occupano decisamente meno spazio (dato che sono dotate di

un minor numero di funzionalità) e permettono di ridurre notevolmente le dimensioni delle applicazioni compilate con esse.

I sorgenti delle uclibc sono stati scaricati dal sito <http://www.uclibc.org/> del suo autore Erik Andersen³³ e con esse i file dimostrativi dell'ambiente per la compilazione della struttura di base denominato “buildroot”.

Dopo numerosi rimaneggiamenti e dopo un'accurata riscrittura delle fasi di compilazione, con annessa integrazione di applicazioni quali *udhcp*, *tftp-client*, *bzip*, *tar*, *bash*, necessarie alla realizzazione dello strato di virtualizzazione, si è raggiunto un primo prototipo che, una volta entrato in funzione, esegue i seguenti passi:

- scarica dal server TFTP lo script contenente le operazioni da eseguire successivamente al caricamento del kernel nel sistema;
- crea un file-system in ram (utilizzando il supporto tmpfs di GNU/Linux);
- monta il file-system di root esportato dal nodo server HPC mediante il protocollo NFS in un'apposita directory;
- scarica i prototipi dei file di configurazione dal nodo server HPC mediante protocollo TFTP;
- ricerca le informazioni specifiche del nodo client HPC facendo il parsing dei parametri di configurazione del protocollo DHCP;
- crea dinamicamente i file di configurazione e li archivia nel file-system creato in ram;
- collega, mediante link simbolici, i file di configurazione archiviati nel file-system caricato in ram nelle locazioni corrette del file-system di root montato attraverso il protocollo NFS;
- esegue un chroot nella locazione dove precedentemente avevamo montato il file-system di root;

³³Erik Andersen <http://codepoet.org/andersen/erik/erik.html>

- avvia il processo di init della distribuzione principale e completa il caricamento della stessa.

L'automatizzazione di questa porzione del boot di sistema ci ha permesso di:

- ridurre lo sforzo amministrativo permettendo di amministrare 5 o 500 nodi senza incrementare il livello di complessità ed il quantitativo di energie necessarie;
- annullare completamente la ridondanza dei file di configurazione sul nodo server HPC;
- rendere possibile l'implementazione di cluster con livelli di complessità normalmente impensabili con l'utilizzo di un numero ridotto di amministratori di sistema.

2.8 Risoluzione dei nomi

La risoluzione dei nomi all'interno dell'intero cluster Beowulf è stata demandata all'utilizzo del file di configurazione */etc/hosts*, permettendo di fatto una riduzione della complessità dell'intera implementazione della risoluzione dei nomi ed aumentando la solidità dell'intero sistema, che non deve dipendere da un servizio esterno e pertanto suscettibile di malfunzionamenti.

Nel caso si decida di incrementare notevolmente il numero dei nodi sarà possibile optare per un servizio di risoluzione dei nomi che utilizzi LDAP oppure un database come backend.

2.9 Installazione del servizio Secure Shell in modalita' shared key

L'implementazione di una struttura autoconfigurante e parzialmente virtualizzata ha suggerito l'integrazione di “servizi di comodo” tipicamente utilizzati in grossi sistemi di elaborazione batch.

Si è pensato, infatti, di utilizzare in modo diretto anche i singoli nodi del cluster, senza necessariamente dover sottomettere i job al sistema, autenticandosi su uno dei due nodi master HPC. Essenzialmente, si è pensato di creare un meccanismo in grado di far entrare l'utente che deve inviare una determinata elaborazione al cluster sempre sul nodo meno carico di sistema.

Da alcune ricerche è emerso che una tale tipologia di servizio, ovvero l'analisi del carico dei nodi e la conseguente redirectione degli utenti sul nodo più scarico, rientra nella sfera dei servizi *Linux Virtual Server* implementati nel kernel di GNU/Linux e pertanto si è deciso di modificare l'immagine root dei nodi client HPC, al fine di condividere una singola chiave SSH³⁴ per l'intero cluster, e si è rimandata la configurazione del sistema di bilanciamento del carico al momento dell'implementazione della porzione ad alta disponibilità del cluster.

2.10 Configurazione delle autenticazioni centralizzate LDAP

Dovendo prevedere un alto numero di utilizzatori ed al fine di semplificare l'archiviazione e la replica dei parametri di autenticazione sul cluster (ricordiamoci che una volta implementata la porzione ad alta disponibilità del cluster avremo due nodi

³⁴ Ssh: acronimo di "secure shell" ed è un elemento della suite di programmi OpenSSL che si occupa di stabilire una connessione sicura, criptata e protetta da password, fra due host.

master HPC ridondati e non un solo nodo master), si è scelto di spostare tutte le autenticazioni degli utenti su un servizio di Directory Service denominato OpenLDAP.

Tale implementazione, completamente basata su software open source e ottimamente integrata nei sistemi GNU/Linux, si compone essenzialmente di due parti:

- una porzione server, assolta dall'ottimo OpenLDAP (<http://www.openldap.org>);
- una porzione client, assolta da due applicazioni create dall'australiana PADL (<http://www.padl.com>), *nss_ldap* e *pam_ldap*.

Per quanto riguarda la porzione server, si è installato il software OpenLDAP sul nodo server HPC e si è provveduto a migrare i dati di autenticazione già creati nel cluster, utilizzando le utility Perl³⁵ *MigrationTools* (anch'esse create da PADL).

Per quanto riguarda la porzione client, si è provveduto ad integrare l'autenticazione LDAP all'interno del meccanismo di autenticazione, utilizzato dalla totalità delle distribuzioni GNU/Linux, ovvero il PAM.

In sostanza l'installazione della porzione client non è stata null'altro che l'installazione del pacchetto software e la configurazione dei dati per il raggiungimento in rete del server OpenLDAP.

Il file di configurazione di OpenLDAP è disponibile nell'appendice.

³⁵ Perl: acronimo di Practical Extraction and Report Language, è un linguaggio di programmazione creato da Larry Wall e tuttora integrato nella totalità delle distribuzioni GNU/Linux.

2.11 Installazione e configurazione di DISTCC e CCACHE per ridurre i tempi di sviluppo

L'ammontare delle ore utilizzate nella compilazione di codice sorgente ha suggerito l'opportunità di sfruttare la potenza di calcolo del cluster per ridurre il tempo di sviluppo e nella fattispecie il tempo di sviluppo dell'intero cluster.

Dopo alcune ricerche attraverso i motori di ricerca, sono emersi due progetti in grado di essere integrati nell'infrastruttura esistente senza risultare troppo invasivi.

Tali software, denominati DISTCC e CCACHE, permettono rispettivamente di distribuire i lavori di compilazione su un insieme di macchine e di ottimizzare la compilazione del software mediante l'utilizzo di cache su disco.

Per quanto riguarda l'installazione di CCACHE, si è provveduto unicamente a compilare ed installare il relativo pacchetto utilizzando il comando:

```
emerge CCACHE
```

ed aggiornare la variabile d'ambiente contenente il PATH dei binari del sistema, mettendo come prima voce la directory contenente i binari di CCACHE. Tale operazione ci ha permesso di “mascherare” i binari del compilatore con appositi script di sistema che automaticamente caricano il binario di CCACHE passando ad esso le opzioni di compilazione che normalmente verrebbero passate al compilatore stesso.

Ad esempio, se normalmente un programma in fase di compilazione esegue un'istruzione di questo tipo:

```
i386-pc-linux-gnu-gcc -I.. -c main.c
```

dopo l'installazione di CCACHE verrà chiamato prima di tutto lo script fittizio e successivamente il binario di CCACHE stesso provvederà a chiamare il binario del compilatore, pertanto, un'istruzione come quella sopra citata corrisponderà a questa:

```
ccache -I.. -c main.c
```

Per quanto riguarda l'installazione di DISTCC, si è provveduto a compilare ed installare il pacchetto utilizzando il comando:

```
emerge distcc
```

successivamente, si è utilizzato il tool *distcc-config* per istruire il sistema sui nodi da utilizzare per la compilazione distribuita. Per finire, si è modificata la variabile d'ambiente PATH attribuendo una priorità maggiore ai binari di DISTCC rispetto ai binari di sistema e si è avviato il demone per lo scambio dei job di compilazione.

2.12 Installazione e configurazione di MPI

Dal momento che la struttura base dell'intero cluster beowulf è ormai abbozzata, si è provveduto ad installare la libreria di message passing, denominata mpich, giunta alla versione 1.2.5.2 .

La compilazione e l'installazione è stata effettuata lanciando l'usuale comando:

```
emerge mpich
```

sia sul nodo master HPC sia su uno dei nodi slave HPC. Successivamente, si è provveduto a creare un file denominato */usr/share/mpich/machine.LINUX* contenente i nomi dei nodi dell'intero cluster beowulf coadiuvati dal numero di CPU presenti sul nodo stesso:

```
slave1:2  
slave2:2  
...
```

Inoltre, si è provveduto a modificare le variabili d'ambiente sia sul nodo master HPC, sia sul nodo slave HPC, nella fattispecie è stata aggiunta la seguente riga nel file */etc/profile*:

```
export P4_GLOBSIZE=200000000
```

Tale modifica ci permette di incrementare la dimensione massima dei pacchetti di MPI a circa 200 MB, che verranno poi convertiti dalla libreria stessa in pacchetti di dimensioni adeguate per essere trasmessi mediante protocollo TCP. Questo valore è stato precedentemente verificato in elaborazioni quali quella che verrà effettuata nel nostro cluster, pertanto, non corriamo l'errore di aver sottodimensionato oppure sovradimensionato i parametri operativi.

Per finire, si è provveduto a modificare, su tutti i nodi del cluster, il file di configurazione */etc/host.equiv* per evitare di dover ridigitare la password dell'utente ogni qual volta l'infrastruttura MPI lanci dei comandi rexec o rsh su nodi remoti.

2.13 Risoluzione delle problematiche di sicurezza del Cluster

Come precedentemente illustrato, l'intero cluster risulta accessibile dalla rete esterna unicamente facendo transitare le comunicazioni attraverso l'indirizzo IP pubblico di uno dei due nodi master HPC, pertanto, in via cautelativa, si è deciso di migliorare il livello di sicurezza dell'intero sistema agendo su due diversi livelli:

- filtraggio del traffico in entrata dalle interfacce pubbliche;
- impostazione delle politiche di tcp wrapping nei due nodi master HPC.

Il filtraggio del traffico è stato realizzato sfruttando l'ottimo tool *iptables*, fornito con la distribuzione Gentoo Linux. In particolare si è deciso di filtrare ogni comunicazione in entrata sull'interfaccia pubblica escluse quelle dirette verso la porta 22, normalmente utilizzata dalle comunicazioni SSH (Secure Shell) intrinsecamente sicure.

Le impostazioni del sistema tcp-wrapper, invece, sono state effettuate modificando i files */etc/hosts.allow* ed */etc/hosts.deny*, permettendo l'accesso ai servizi di rete unicamente ai nodi del cluster beowulf oppure ai client autorizzati. Solamente il servizio *sshd* è stato lasciato svincolato dal sistema di *tcp wrapping*.

CAPITOLO 3

HPC A 64 BIT (AMD64 OPTERON)

Durante il presente lavoro di tesi è stato possibile provare, per un periodo limitato di tempo, una macchina basata su tecnologia AMD64 Opteron, pertanto, si è provveduto ad integrarla nel cluster Beowulf e si è provveduto a fare alcuni test prestazionali per quantificare l'aumento di performance derivante dall'adozione di una simile tecnologia in un sistema di supercalcolo.

3.1 Introduzione alla tecnologia AMD64

L'introduzione, da parte di AMD, delle CPU con tecnologia AMD64 Opteron nel mercato consumer ha tracciato la nuova rotta dell'informatica personale in ambito x86.

Tali CPU, oltre a rappresentare il primo vero tentativo di costruire CPU a 64bit con tecnologia CISC (i processori RISC a 64bit esistono da anni), introducono una nutrita schiera di vantaggi tecnici che ne stanno determinando il successo:

- raddoppio della capacità d'indirizzamento della memoria con conseguenti benefici per quelle applicazioni che fanno un utilizzo pesante delle richieste verso la memoria stessa;
- raddoppio del numero di connessioni tra la memoria ed i dispositivi periferici con conseguente raddoppio della banda di I/O³⁶ della CPU;
- compatibilità nativa con il codice a 32bit e nessun conseguente degrado delle performance nell'esecuzione di codice preesistente;

³⁶ I/O: Forma abbreviata per indicare Input-Output. Generalmente si riferisce a operazioni eseguite sulle memorie coinvolgenti flussi di dati in input o in output.

- facile integrabilità nei progetti hardware esistenti, data la presenza di un controller di memoria integrato nel chip che riduce notevolmente la complessità dell'intero sistema;
- bassissimi consumi derivanti dalla totale riprogettazione del core principale della CPU.

In passato anche Intel ha cercato di introdurre una CPU CISC a 64bit, denominata Itanium IA-64, tuttavia, i costi eccessivi, l'incompatibilità con il codice esistente e le basse performance dell'intera soluzione hanno relegato tale tecnologia in settori di nicchia dell'information technology.

3.2 Analisi dell'hardware

La macchina utilizzata per i test è un dual AMD Opteron 140 con 2GB di ram DDR e due dischi SATA da 250GB in configurazione RAID³⁷ 0. L'intero sistema è ospitato all'interno di un case formato tower ATX ed è coadiuvato da un potente alimentatore da 500W che assicura un'adeguata riserva energetica.

Come prima cosa si è provveduto a scaricare un'immagine ISO contenente Gentoo Linux a 64bit, successivamente, si è avviato il sistema mini-ambiente d'installazione per verificare il supporto dell'hardware del sistema in oggetto.

Da una prima analisi è emerso che il chip del controller SATA utilizzato nel sistema era indicato come parzialmente funzionante / instabile nelle note tecniche dell'installazione, pertanto, si è optato per una sostituzione del disco SATA con un disco in tecnologia EIDE ATA 133.

³⁷ RAID: Acronimo di Redundant Array of Inexpensive Disk. Tale tecnologia, riassunta in vari livelli, permette di utilizzare un gruppo di dischi al fine di creare un device virtuale in grado di fornire livelli di performance più elevati oppure un livello superiore di affidabilità e pertanto una tolleranza maggior ai guasti dei dischi.

Una volta completata la sostituzione del disco, si è provveduto a riavviare l'ambiente di installazione e procedere al partizionamento del disco fisso secondo il seguente schema:

Nome partizione	Dimensione (Mb)	Mount point	Tipo	Opzioni
hda1	100	/boot	ext2	noauto
hda2	1000		swap	
hda3	39000	/	ext3	noatime

Tabella 3.1: Schema di partizionamento

Dopo aver creato le partizioni secondo lo schema indicato, si è provveduto a creare i file-system su ogni partizione, in particolare:

mkswap /dev/hda2 per la partizione di swap

mkfs.ext3 /dev/hda1 per la partizione /boot

mkfs.ext3 /dev/hda3 per la partizione /

Fatto questo si è provveduto ad attivare la partizione di swap col comando *swapon /dev/hda2* e a montare le diverse partizioni sui rispettivi mount point con il comando *mount /dev/hdax /mnt/gentoo/path*, dove */path* corrisponde al percorso relativo a ogni partizione come specificato nella tabella XXX.

Succesivamente, analogamente a quanto è stato fatto in fase di installazione dei nodi master HPC e slave HPC, si è provveduto a:

- configurare la rete attraverso il comando *ifconfig*;
- entrare nella partizione di root attraverso il comando:

```
cd /mnt/gentoo
```

- scompattare lo stage3 presente nel cd di installazione:
`tar xvjf /mnt/cdrom/stages/stage3*`
- scaricare l'elenco dei mirror dei sorgenti dei pacchetti:
`mirrorselect -a -s4 -o >> /mnt/gentoo/etc/make.conf`
- montare il file-system speciale /proc: `mount -t proc none /mnt/gentoo/proc;`
- creare un file `/mnt/gentoo/etc/resolv.conf` contenente gli indirizzi dei server DNS da utilizzare durante gli scaricamenti da internet;
- montare il file-system speciale /dev:
`mount -t devfs none /mnt/gentoo/proc`
- entrare nell'ambiente chroot dello stage3:
`chroot /mnt/gentoo`
- installare i sorgenti del kernel:
`emerge gentoo-sources`
- installare l'utilità per la compilazione del kernel:
`emerge gentoolkit`
- installare il boot loader:
`emerge grub`

A questo punto, dal momento che la radice di sistema è diventata `/mnt/gentoo`, si sono eseguite una serie di operazioni per aggiornare l'ambiente:

```
env-update
source /etc/profile
```

e si è lanciato l'aggiornamento dell'indice dei pacchetti:

```
emerge rsync
```

Una volta completata la generazione dell'indice aggiornato dei pacchetti, si sono inserite le seguenti ottimizzazioni di compilazione all'interno del file */etc/make.conf*:

```
CHOST="x86_64-pc-linux-gnu"  
CFLAGS="-march=k8 -O2 -pipe"  
CXXFLAGS="-march=k8 -O2 -pipe"  
MAKEOPTS="-j3"
```

e si è lanciato l'aggiornamento del sistema in modo da avere una distribuzione aggiornata con gli ultimi rilasci del software, pertanto si è lanciato il seguente comando:

```
emerge -uD world
```

Tale procedura ha richiesto, seppur su una macchina potente, oltre dieci ore di elaborazione ininterrotta.

3.3 Configurazione ed Installazione di un kernel a 64bit

Una volta completati i passi iniziali per l'installazione e per l'aggiornamento dell'ambiente base della distribuzione Gentoo Linux a 64bit, si è proceduto con la configurazione ed installazione del kernel a 64bit, nella fattispecie si è utilizzato il comando:

```
gentoolkit -menuconfig all
```

che permette di:

- entrare nei sorgenti del kernel;
- pulire i residui di eventuali compilazioni precedenti;
- entrare nel menù di configurazione del kernel e scegliere i moduli dello stesso;
- installare il kernel ed i moduli nelle directory del sistema.

Successivamente, si è installato il boot loader nel Master Boot Record dell'hard disk e si è provveduto ad inserire la voce indicante il kernel appena installato all'interno del file di configurazione dello stesso.

Al termine di questa operazione si è riavviato il sistema.

3.4 Adattamento delle distribuzioni linux alla nuova architettura AMD64

L'utilizzo del sistema ha rivelato delle notevoli potenzialità della nuova architettura di AMD. In particolar modo si è potuta apprezzare una snellezza del sistema senza precedenti. L'intera struttura sembra essere estremamente fluida e sempre in grado di gestire pesanti carichi di lavoro. Inoltre l'adozione di bus dati indipendenti ha permesso di sfruttare appieno dei sottosistemi quali la scheda di rete gigabit, il sottosistema a dischi e la RAM.

Alcuni test, eseguiti in modo continuativo nel corso di un'intera settimana, hanno dimostrato un'eccellente stabilità dell'intero sistema che lo rende un ottimo candidato per la creazione di un cluster per il calcolo ad alta performance.

L'unico neo è rappresentato dalla scarsità delle applicazioni predisposte per il funzionamento a 64bit. Questo è dovuto in primo luogo al rilascio recente delle CPU

basate su architettura AMD64 ed in secondo luogo alla massiccia diffusione di GNU/Linux in ambienti con una prevalenza di architetture x86 a 32bit.

Per sopperire, almeno in parte, alla mancanza di alcune applicazioni è stato intrapreso un minuzioso lavoro di testing ed adattamento di alcuni dei pacchetti forniti da Gentoo Linux ed in un buon numero di casi si è riusciti a colmare tale lacuna.

3.5 Ottimizzazione e test prestazionali su compilatori di nuova generazione (GCC-3.4.x)

Da alcune ricerche, riguardanti le ottimizzazioni del codice su piattaforme basate su tecnologia AMD64, è emerso come il team di sviluppo del compilatore GNU/Linux gcc stia integrando un notevole quantitativo di ottimizzazioni capaci di incrementare massicciamente le performance del codice compilato su architetture AMD64.

Allo stato attuale i compilatori contenenti tali ottimizzazioni sono da considerarsi in fase di testing, tuttavia, tali premesse depongono sicuramente a favore della nuova architettura, che, seppur già velocissima, potrà diventare ancor più veloce una volta sfruttata appieno.

CAPITOLO 4

HA

All'interno del presente capitolo verrà analizzata la realizzazione di un'infrastruttura ad alta disponibilità in grado di estendere le funzionalità tipiche di un sistema di supercalcolo, permettendo di ottenere livelli di continuità del servizio (uptime) molto elevati.

Per comprendere appieno la caratura della soluzione tecnica adottata, è necessario definire minuziosamente il significato di alta disponibilità evitando ogni possibile fraintendimento; partiremo pertanto da una definizione generica per giungere infine ad una definizione più rigorosa.

Prima di tutto possiamo affermare che un sistema HA, nella sua “incarnazione” più generica, è un sistema in grado di dotare i servizi della cosiddetta “fault tolerance”, ovvero ci consente di mascherare eventuali guasti hardware in modo che non vengano percepiti dall'utente.

Tale funzionalità, inizialmente integrata in sistemi progettati per gestire servizi critici, ora, grazie alla notevole riduzione dei costi ed alla presenza di molti progetti open source di alta qualità (alcuni dei quali utilizzati in questa tesi), viene integrata anche in comunissimi server e viene proposta anche per “supportare” servizi meno vitali per l'infrastruttura informatica.

Come precedentemente accennato, normalmente i sistemi HA sono antagonisti dei sistemi HPC, tuttavia, nel nostro caso, dovendo applicare il supercalcolo ad applicazioni mediche, si è dovuto cercare un punto di convergenza delle due tecnologie.

4.1 Livelli di disponibilità

La disponibilità di un servizio può essere suddivisa in quattro livelli, ognuno dei quali include il precedente, che possono essere schematizzati come segue:

<i>Livello di disponibilità</i>	<i>Descrizione</i>
Livello 1 - Disponibilità normale	<p>Rappresenta il livello minimo di disponibilità. L'unica protezione contro eventuali interruzioni nell'erogazione del servizio è il backup dei dati, eseguito secondo una politica decisa dall'amministratore di sistema.</p> <p>Risulta evidente che in caso di guasto e successivo ripristino del sistema potrebbe essere possibile ripristinare solo in parte i dati, che risulterebbero aggiornati in modo relativo (in funzione della frequenza dei backup).</p> <p>E' altresì ovvio che l'erogazione del servizio potrà essere ripristinata unicamente solo quando l'amministratore di sistema avrà completato il ripristino dell'intero sistema e riterrà opportuno riprendere l'erogazione del servizio.</p> <p>Pertanto, il tempo di ripresa dell'attività risulta legato anche ad un fattore umano e da attente analisi della problematica emerge che in situazioni critiche, quale il ripristino di un sistema informativo, l'incidenza dell'errore umano aumenta in modo esponenziale.</p>

<i>Livello di disponibilità</i>	<i>Descrizione</i>
Livello 2: Disponibilità maggiore	<p>Lo scopo di un sistema in grado di garantire questo livello di disponibilità è quello di garantire un miglior livello di aggiornamento dei dati anche in caso di guasto di uno o più dischi (a seconda dalla tecnologia utilizzata).</p> <p>Tale obiettivo è raggiunto grazie alla replicazione fisica dei dati su più dischi attraverso tecnologie di RAID.</p> <p>Anche in questo caso è possibile mantenere i dati aggiornati anche nell'eventualità di danni lievi ai dispositivi di memorizzazione di massa. Tuttavia, la ripresa dell'erogazione piena del servizio dipenderà, almeno parzialmente, dalla disponibilità dell'amministratore di sistema.</p>

<i>Livello di disponibilità</i>	<i>Descrizione</i>
Livello 3: Alta disponibilità.	<p>Questo livello garantisce la protezione del sistema e dell'erogazione dei servizi anche in caso di guasto di una porzione significativa del sistema.</p> <p>Risulta evidente come questo livello rappresenti un notevole salto di qualità rispetto ai livelli precedenti. In soluzioni di questo tipo non siamo più dipendenti dal livello di disponibilità dell'amministratore di sistema e soprattutto la ripresa dell'erogazione del servizio non è più vincolata ad eventuali errori umani.</p> <p>Un sistema siffatto è costituito da un cluster di due nodi configurati in modo tale che il secondo prenda il posto del primo nel caso in cui quest'ultimo non sia più in grado di erogare il servizio (sia in caso di rottura di un componente hardware sia in caso di manutenzione programmata del sistema stesso).</p>

<i>Livello di disponibilità</i>	<i>Descrizione</i>
Livello 4: Disaster Recovery.	<p>Questo livello, che rappresenta l'evoluzione dell'alta disponibilità, permette di mettere al riparo i nostri dati anche da eventi che comportino la distruzione dell'intero sito nel quale sono collocate le macchine.</p> <p>Essenzialmente, si procede con una replicazione dell'intera infrastruttura e dei dati in essa contenuti, in un'area geografica più o meno distante in funzione del livello di disponibilità che si vuole ottenere.</p>

Tabella 4.1: Livelli di disponibilità

Per meglio inquadrare l'importanza dell'implementazione dei vari livelli di disponibilità del servizio, si può osservare il grafico seguente che riporta una statistica calcolata dall'IEEE, risalente all'Aprile del 1995, che mostra le classiche cause di downtime dei sistemi di elaborazione.

Cause di Downtime

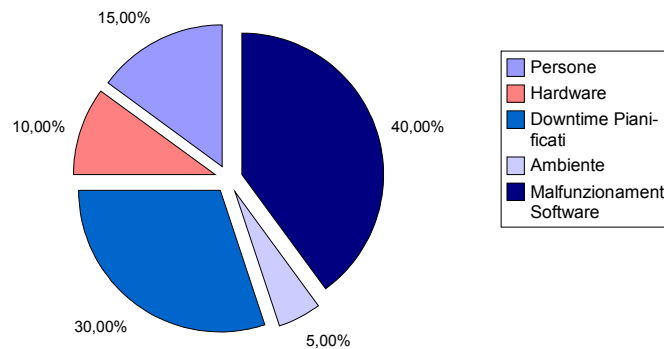


Figura 4.1: Cause di downtime

Come si evince dal grafico, la causa principale è il malfunzionamento software che per sua natura non è prevedibile dato che riguarda errori di programmazione che si presentano solamente in certe condizioni non note a priori. Pertanto, risulta evidente come sia difficile fornire un servizio qualitativamente valido quando abbiamo una probabilità di 4 su 10 che si verifichi un malfunzionamento.

4.2 Il requisito dei cinque 9

Un sistema ad alta disponibilità tipicamente viene classificato in base a tre fattori:

- tempo di disponibilità del servizio;
- prestazioni globali del sistema;

- costo dell'intera implementazione.

E' evidente come il tempo di disponibilità del servizio sia quello che maggiormente caratterizza un sistema ad alta disponibilità. Pertanto, uno dei parametri presi generalmente in considerazione è il tempo di disponibilità garantita definita come il rapporto percentuale fra il tempo di erogazione effettiva del servizio ed il tempo di osservazione.

L'obiettivo ideale di ogni sistema ad alta disponibilità è quello di raggiungere la fatidica soglia dei cinque 9, ovvero un tempo di erogazione effettiva del servizio pari al 99,999% del tempo di osservazione.

Se consideriamo un tempo di osservazione di un anno solare, troviamo facilmente che il tempo massimo consentito per la sospensione del servizio è di soli 5 minuti e 15 secondi, che equivalgono a 6 secondi per settimana. Tale valore è alquanto difficile da ottenere e fino a pochi anni fa era raggiungibile solamente con implementazioni ad appannaggio di grandi aziende specializzate nella creazione di sistemi ad alta disponibilità.

Con la diffusione del software open source ed in particolare con la diffusione del sistema operativo GNU/Linux, sono stati creati diversi software in grado di avvicinarsi alla fatidica soglia del 99,999% di uptime del servizio. I due più conosciuti sono keepalived (<http://www.keepalived.org>) e Linux-HA (<http://www.linux-ha.org>). All'intero del presente lavoro di tesi è stato utilizzato Linux-HA in quanto più personalizzabile.

I tempi di sospensione consentiti a seconda della percentuale di erogazione sono riassunti nella seguente tabella:

Percentuale Uptime	Downtime per anno	Downtime per settimana
98,00%	7,3 giorni	3 ore, 22 minuti
99,00%	3,65 giorni	1 ora, 41 minuti
99,80%	17 ore, 30 minuti	20 minuti, 10 secondi
99,90%	8 ore, 45 minuti	1 10 minuti, 5 secondi
99,99%	52 minuti 30 secondi	1 minuto
99,999%	5 minuti, 15 secondi	6 secondi
99,9999%	30,5 secondi	0,6 secondi

Tabella 4.2: Tempi di sospensione in base alla percentuale di erogazione

Per ottenere percentuali di uptime sempre più elevate, è necessario analizzare nel dettaglio l'infrastruttura che si vuole realizzare ed eliminare progressivamente ogni “single point of failure”, ovvero tutti gli elementi che smettendo di funzionare regolarmente possono portare ad un collasso dell'intera infrastruttura, interrompendo pertanto l'erogazione del servizio stesso.

Le possibili soluzioni a questo problema sono essenzialmente di due tipi:

1. introduzione della ridondanza nell'hardware chiave del sistema;
2. l'adozione di soluzioni clusterizzate.

L'introduzione di componenti ridondanti nel sistema, sebbene migliori il livello di disponibilità medio del servizio, non ci pone al riparo da diverse tipologie di guasto ed implica, se estremizzata, un incremento smisurato dei costi di realizzazione.

Pertanto fino a quando introduciamo la ridondanza in componenti relativamente economici quali le schede di rete, gli hard disk, gli alimentatori, possiamo arrivare ad avere consistenti vantaggi in termini di disponibilità con un aggravio in termini di costi pari al 1-2% del costo totale del sistema, mentre se cerchiamo di ridondare componenti chiave quali le CPU, i moduli di memoria RAM, i controller RAID del sistema, ecc, i costi possono lievitare fino al 300-400% del costo del sistema base.

Come si può facilmente dedurre, un sistema siffatto diventa, pertanto, poco conveniente in termini economici ed induce alla ricerca di implementazioni alternative in grado di offrire gli stessi livelli di disponibilità a costi decisamente più ridotti.

Con il passare degli anni e con l'evoluzione delle tecniche di programmazione del software e dei sistemi operativi, si sono creati programmi in grado di trasformare una collezione di due o più personal computer in un server virtuale in grado di erogare i servizi con livelli di continuità compatibili con la faticosa soglia dei cinque 9.

Tale tecnica prende il nome di clustering HA (High Availability) ed attualmente è talmente diffusa da essere presente in ogni reparto dell'information technology. Aziende quali Google, Porsche, Sony, Motorola, FedEx Inc., MAN, ecc., stanno utilizzando da anni sistemi basati sulla tecnologia Linux Virtual Server (ed in particolare Linux-HA) all'interno di diverse sezioni critiche della loro infrastruttura informatica.

Probabilmente il segreto di tale diffusione è da ricercarsi anche nella semplicità dell'idea che sta alla base di tale soluzione ad alta disponibilità. Nell'implementazione più semplice abbiamo due macchine gemelle che sono in grado di erogare intercambiabilmente un determinato insieme di servizi e che sono dotate di un software di analisi dello stato dei servizi in grado di commutare l'erogazione del servizio stesso da un server ad un altro. Pertanto, se una delle macchine viene danneggiata mentre sta erogando un servizio, ad esempio dalla rottura di un componente hardware, il sistema

automaticamente provvede a spostare l'erogazione del servizio sul server superstite, escludendo dall'intera infrastruttura la macchina danneggiata attraverso uno spegnimento controllato.

Delle tecniche di “spoofing” in grado di “nascondere” alla rete la migrazione del servizio da un server ad un altro completano il pacchetto di funzionalità.

L'affinamento di questi passaggi ed alcune accortezze in termini di implementazione del servizio permettono di non far percepire il guasto all'utente che non noterà alcuna discontinuità nell'erogazione del servizio.

Inoltre, tale soluzione presenta il grosso vantaggio di poter utilizzare hardware generico e quindi non particolarmente oneroso in termini di costi.

Nel nostro caso, ad esempio, per la creazione della porzione HA sono state utilizzate due macchine di recupero dalle caratteristiche hardware e dalle performance decisamente modeste.

4.3 Due Geometrie di clustering HA: Active-Active ed Active-Standby

Quando ci si accinge a realizzare un cluster per l'alta disponibilità è molto importante decidere esattamente quali servizi dovranno erogare i singoli nodi del sistema.

Principalmente si distinguono due tipi di implementazioni:

- active-active: caratterizzata dal fatto che ogni nodo del cluster eroga in condizioni normali uno o più servizi dell'intero cluster HA;
- active-standby: caratterizzata dal fatto che contemporaneamente avremo sempre e solo un nodo erogante attivamente i servizi ed uno o più nodi dormienti in attesa di prendere il posto del server attivo in caso di necessità.

4.4 Creazione di master ridondanti

Per la creazione dei master ridondanti, per prima cosa, si è proceduto con la replica del primo nodo master HPC sulla macchina gemella (come precedentemente accennato i due nodi master HPC sono basati su tecnologia Intel Pentium 3, mentre i nodi slave HPC sono basati su tecnologia AMD Athlon MP).

In secondo luogo, si è proceduto a pianificare una serie di interventi volti all'integrazione dei servizi per la gestione dell'alta disponibilità con i servizi presenti sulla macchina.

A prima vista creare un nodo master HPC ridondante potrebbe sembrare uno spreco di risorse, dato che viene “sprecata” della potenza di calcolo impiegabile per incrementare le performance dell'intero sistema distribuito, ma si pensi al danno in termini economici e di produttività derivante da un eventuale fermo di tutto il sistema fino alla risoluzione dell'eventuale guasto hardware. Ovviamente, l'incidenza del costo di un sistema dotato di alcuni elementi ridondanti si riduce notevolmente con la crescita del numero di nodi del sistema distribuito stesso. Nel nostro caso, l'incidenza totale dei costi, se paragonata al costo del singolo nodo slave HPC, è risultata essere ben poca cosa rispetto all'incremento sostanzioso della disponibilità dell'intero sistema che si è ottenuto.

4.5 Abilitazione del supporto LVS nei due nodi master HPC

Durante le indagini iniziali, legate al reperimento di documentazione ed al delineamento della struttura del sistema distribuito oggetto di questa tesi, è emerso come molti tool open source fossero disponibili in versioni “di produzione” unicamente su kernel appartenente alla serie 2.4.x, pertanto, sui nodi master HPC (che d'ora in avanti possono

essere considerati anche director³⁸ HA) si è deciso di utilizzare un kernel 2.4.26 al posto dei recenti 2.6.x.

L'unico svantaggio derivante dall'utilizzo di un kernel appartenente al vecchio tree di sviluppo è stato quello di dover aggiungere manualmente le funzionalità LVS, non ancora inserite ufficialmente nei sorgenti distribuiti attraverso <http://www.kernel.org>.

Pertanto si è provveduto a compiere i seguenti passi:

- scaricare dal sito <http://www.linux-ha.org> l'ultima versione delle estensioni LVS per il kernel in uso;
- decompattare i sorgenti del kernel;
- decompattare i sorgenti delle utility e dei patch del kernel per il supporto LVS;
- inserire il supporto LVS nel kernel di sistema tramite l'utility *patch*;
- attivare il supporto LVS nella configurazione del kernel;
- ricompilare ed installare il kernel stesso nel sistema;
- duplicare il kernel installato nel secondo nodo master HPC.

4.6 Installazione e configurazione di Heartbeat per la gestione delle problematiche di HA all'interno del cluster

Completata l'installazione del supporto LVS nel kernel dei due nodi master HPC, si è provveduto ad installare il software linux-HA con un set di utility atte a configurare un efficace ambiente ad alta disponibilità.

³⁸ Director HA: un computer agente da gateway, dotato di funzionalità LVS.

Pertanto si è provveduto ad eseguire i comandi:

```
emerge ipvsadm  
emerge heartbeat  
emerge iproute2
```

Una volta completata l'installazione, si è proceduto alla configurazione del demone heartbeat, responsabile del monitoring dello stato dei servizi installati sui due nodi master HPC ed altresì responsabile della migrazione del pacchetto dei servizi da un server ad un altro in caso di malfunzionamento.

Nello specifico si è intervenuti su tre aree distinte:

- configurazione del demone heartbeat;
- configurazione dell'autenticazione del servizio d'interscambio delle informazioni di stato;
- configurazione del gruppo di servizi allocati sui server.

Per quanto concerne la configurazione del demone heartbeat (visionabile nell'appendice) si è provveduto a configurare i seguenti parametri:

- i mezzi trasmissivi utilizzati per le comunicazioni delle informazioni di stato dei nodi master HPC che sono il cavo null-modem ed il cavo ethernet (collegati tra le schede di rete dei due nodi master HPC / director HA);
- il numero di tentativi di connessione al nodo partner prima di considerarlo come “decaduto” e pertanto iniziare la migrazione dei suoi servizi sul nodo superstite;
- il numero di secondi necessari ad un cluster HA per eseguire i controlli iniziali e pertanto iniziare il ciclo continuo di test per determinare l'attività dei nodi;

- il file di log nel quale conservare le informazioni di stato dell'intero sistema ad alta disponibilità;
- la priorità del processo heartbeat (normale oppure realtime);
- l'eventuale connessione con l'interfaccia watchdog del kernel di sistema.

Per quanto concerne l'autenticazione dei demoni heartbeat, si è deciso, dal momento che i due nodi comunicano tra loro tramite un cavo di rete di tipo “crossed” e pertanto non inviano le informazioni di stato tramite una rete pubblica, di utilizzare un'autenticazione basata su una semplice password condivisa.

In fine, si è provveduto a configurare la distribuzione dei servizi sui due server in modo da realizzare una configurazione active-standby pura, ovvero composta da un server attivo configurato per fornire tutti i servizi necessari al nostro cluster HPC ed un server dormiente in attesa di prendere il posto del server principale in caso di manutenzione programmata oppure rottura hardware.

4.7 Sincronizzazione delle mappe IPVS tra i directors

Al fine di minimizzare il tempo d'intervento del server in standby e ridurre al minimo il momento di vuoto conseguente alla migrazione dei servizi dal server attivo a quello passivo, si è deciso di dotare entrambi i director di un servizio per l'interscambio delle informazioni di routing tra i nodi.

Tale servizio, derivante dall'implementazione LVS, permette di sincronizzare in tempo reale i dati delle connessioni attive del cluster, permettendo al nodo passivo di continuare ad erogare i servizi senza dover interrompere le connessioni e senza dover costringere i client a richiedere nuovamente una connessione e pertanto ricominciare ogni elaborazione dal principio.

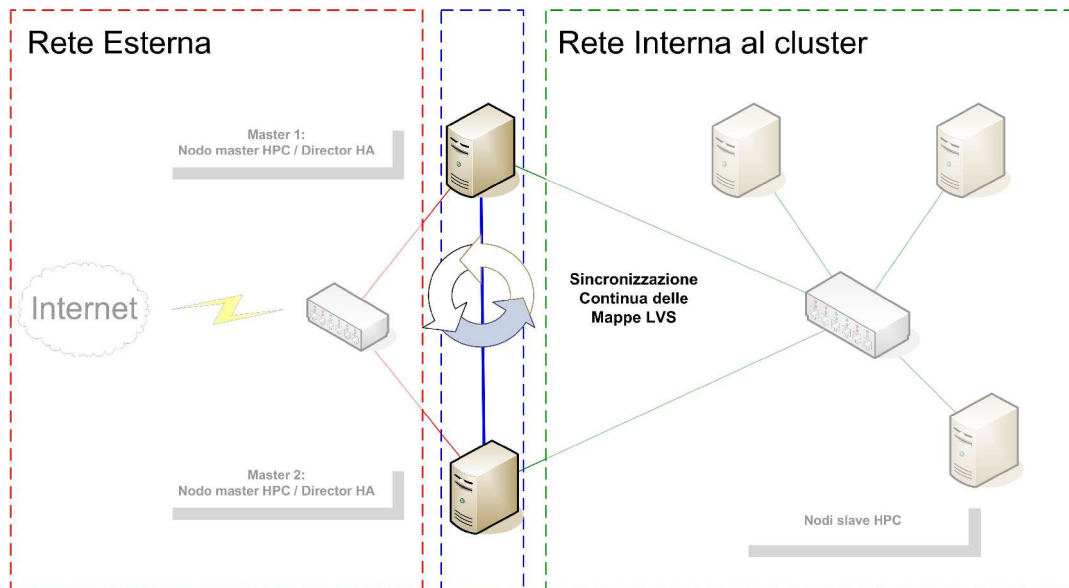


Figura 4.2: Sincronizzazione mappe LVS

4.8 Adattamento dei servizi di rete al supporto di HA

Il servizio heartbeat è stato creato per utilizzare i normali script di sistema, avviare o terminare i servizi da erogare. Tuttavia, poiché si è verificato sperimentalmente come molti script presenti nelle distribuzioni GNU/Linux non tengano minimamente conto di problematiche legate alla loro collocazione in un ambiente clusterizzato, si è resa necessaria una completa riscrittura di tutti gli script utilizzati per gestire i servizi erogati dai nodi master HPC / director HA.

In particolar modo, si è dovuti intervenire pesantemente sullo script per la gestione di NFS al fine di evitare strani fenomeni di disconnessione e riconnessione dei nodi diskless ai nodi master HPC in caso di rottura del master principale e conseguente migrazione del servizio sul master secondario.

Tale opera di minuziosa ricostruzione degli script ha permesso di arrivare ad ottenere un sistema funzionante all'unisono e capace di non interrompere l'esecuzione delle elaborazioni anche dopo uno spegnimento brutale (ovvero disconnessione fisica del cavo di alimentazione) del master attivo.

Nell'appendice è visionabile la copia completa di tali script.

4.9 Aggiunta del supporto failover a DHCP

L'analisi degli archivi della mailing list degli sviluppatori e degli utilizzatori del servizio ISC DHCP ha segnalato la presenza di un supporto di alta disponibilità nelle versioni successive alla 3.0.0. Pertanto, si è provveduto ad estendere le funzionalità del servizio installato sul nodo master HPC, utilizzato come modello per la realizzazione della coppia di server in alta disponibilità.

Nella fattispecie si è aggiunta una sezione in grado di definire il cosiddetto gruppo di server DHCP.

```
server-identifier dhcp1.fn.csr.unibo.it;
...
failover peer "dhcp" {
    primary;
    address 10.0.8.1;
    port 519;
    peer address 10.0.8.2;
    peer port 519;
    max-response-delay 30;
    max-unacked-updates 10;
```

```
mclt 3600;  
split 128;  
load balance max seconds 3;  
}
```

4. 10 Abilitazione delle repliche in OpenLDAP

Analogamente a quanto avvenuto nel caso del demone DHCP, si è provveduto a modificare la configurazione del servizio OpenLDAP, responsabile delle autenticazioni dell'intero cluster, in modo da supportare il servizio di replica.

In sostanza non si è fatto altro che:

- creare un utente *replica* abilitato a leggere e scrivere l'intero albero LDAP;
- creare sul server principale un *log file* per l'archiviazione delle modifiche al database;
- sincronizzare i database OpenLDAP tramite una copia a basso livello degli stessi;
- inserire i riferimenti per permettere al demone *slapd* ed in particolare alla sua controparte *slurpd* (creata per gestire la replicazione dei dati) di inviare ogni modifica effettuata sull'albero principale direttamente al server gemello, permettendo a questo di replicarla sul proprio database.

La copia dei file di configurazione è disponibile presso l'appendice.

4. 11 Replicazione del servizio di risoluzione dei nomi

Data la staticità dei record contenenti i dati per la risoluzione dei nomi dei nodi del cluster in indirizzi IP, si è deciso di installare un servizio di risoluzione dei nomi molto

essenziale e di procedere alla replicazione dei dati tra i due nodi master HPC/director HA semplicemente utilizzando le funzionalità di copia via rete dell'utility *rsync*.

Pertanto, si è provveduto ad installare il demone *djbdns* e l'utilità *rsync* con il consueto comando *emerge* e successivamente non si è fatto altro che:

- inserire gli indirizzi IP ed i nomi dei nodi del cluster HPC all'interno del database di *djbdns*;
- lanciare ad intervalli regolari tramite l'utility *cron* la sincronizzazione dei database:

```
rsync -e ssh -az /etc/tinydns-internal/root/data  
master2:/etc/tinydns-internal/root/data.
```

4.12 Analisi delle soluzioni per la replicazione dei dati tra i due nodi master

Ogni intervento eseguito sul nodo master HPC / director HA è volto ad ottenere una perfetta integrazione tra i servizi necessari al corretto funzionamento di un cluster Beowulf ed i servizi necessari all'alta disponibilità. Pertanto ogni aspetto deve essere rielaborato ed adattato ad una prospettiva globale, ovvero ogni servizio ed ogni funzionalità del singolo nodo master HPC deve essere ripensato in chiave distribuita e quindi deve essere erogabile da entrambi i nodi master HPC / director HA in modo intercambiabile.

Mentre per un servizio “statico” o “pseudo-statico”, quale può essere un DNS, esistono diverse strategie implementative che permettono di arrivare a risolvere brillantemente il problema, quando si cerca di replicare atomicamente i dati contenuti in un filesystem; emergono una moltitudine di problematiche spesso di difficile risoluzione.

Dalle ricerche effettuate in fase di progettazione è emerso che normalmente il problema della replicazione dei dati nei nodi ad alta disponibilità viene risolto in due modi diversi:

- attraverso l'adozione di un sistema a dischi centralizzato, condiviso mediante catena SCSI o Fibre Channel;
- attraverso l'utilizzo di un file-system distribuito sui nodi del cluster HA.

Andiamo ad analizzare brevemente i pro ed i contro di entrambe le soluzioni attraverso una tabella esplicativa.

Unità a dischi esterna

<i>Pro</i>	<i>Contro</i>
Buone performance	Costi elevati
Complessità d'integrazione bassa	Possibile “single point of failure” se non dotato di adeguato livello di ridondanza

Tabella 4.3: Pro e contro dell'unità a dischi esterna

File-System distribuito

<i>Pro</i>	<i>Contro</i>
Costi bassi	Performance ridotte rispetto ad un file-system locale
Nessun “single point of failure”	Spero di spazio dovuto alla ridondanza su ogni nodo del cluster

Tabella 4.4: Pro e contro del File-System distribuito

Data la mancanza di risorse per l'acquisto di un sottosistema a dischi esterno, nel nostro caso è stato scelto di cercare un'alternativa nel gruppo di file-system distribuiti, pertanto sono state analizzate le seguenti alternative:

- file-system distribuito di tipo Coda;
- file-system clusterizzato di tipo OGFS, GFS, OCFS;
- file-system standard (ext3/reiserfs ecc) + Replicated Block Device.

4.13 Distributed Filesystem: Coda

L'indagine sull'utilizzabilità ed integrabilità di Coda all'interno della soluzione ad alta disponibilità oggetto di questa tesi si è interrotta subito dopo l'installazione dello stesso all'interno dei nodi master HPC / director HA.

E' emerso, infatti, come le funzionalità di replicazione pseudo-atomica dei dati fossero molto instabili e prive di un adeguato livello di supporto da parte della comunità degli sviluppatori. Inoltre, si è rilevato come Coda, non avendo un sistema di locking distribuito delle risorse, non sia in grado di garantire l'unicità del dato all'interno del sistema distribuito stesso. Pertanto, due utenti che modificano contemporaneamente una medesima risorsa generano nel sistema due copie distinte dello stesso dato che devono essere fuse manualmente dall'amministratore di sistema. Tale operazione nel peggiore dei casi può addirittura generare un caos totale nella gestione dei dati.

4.14 Clustered filesystem: OGFS, RedHat GFS, Oracle OCFS

Dopo l'esperienza negativa del file-system Coda, si sono ricercati dei file-system distribuiti con funzionalità di clustering (di qui in avanti indicati come file-system clusterizzati) pertanto dotati di un sistema di locking e capaci di dare un accesso esclusivo alla singola risorsa all'utente che ne faccia richiesta.

Nel panorama dei file-system clusterizzati open source sono emersi tre progetti:

- Oracle OCFS: file-system clusterizzato utilizzato da Oracle in abbinamento al database Oracle;
- OGFS: evoluzione del tree open source di sistema GFS;
- RedHat GFS: versione open source del codice di sistema GFS 6.0.

Tuttavia, i tre progetti si sono rivelati non adatti ad essere integrati nel progetto oggetto di questa tesi.

L'Oracle Cluster File System (OCFS) è stato scartato immediatamente in quanto non adatto ad archiviare nulla che non sia il database Oracle.

L'Open Global File System (OGFS) è stato scartato dopo circa due giorni di utilizzo a causa delle innumerevoli instabilità del codice disponibile unicamente in versione di sviluppo.

Il RedHat Global File System (GFS) è stato scartato data l'incompatibilità dei sorgenti disponibili attraverso il sito RedHat Cluster - Global File System con il kernel in uso all'intero dei nodi master HPC / director HA.

In particolare, è stato rilevato come il codice RedHat fosse stato pesantemente rielaborato per integrare delle funzionalità presenti unicamente in kernel di sviluppo. Tali modifiche rendevano praticamente impossibile l'integrazione nei kernel installati.

4.15 Distributed Replicated Block Device: DRBD

A seguito dei tentativi falliti di integrazione di un file-system clusterizzato open source all'interno del cluster, a causa delle ragioni sopra menzionate, si è deciso di adottare un approccio più convenzionale e si è deciso di utilizzare un file-system convenzionale in abbinamento ad un sistema di replicazione atomica dei block device.

Tra le varie alternative disponibili, ovvero ENBD³⁹+MD⁴⁰ e DRBD⁴¹, si è deciso di utilizzare quest'ultima data la sua maturità e date le sue spiccate doti d'integrazione in ambienti ad alta disponibilità.

In sostanza tale software, creato da Philipp Reisner⁴², permette, attraverso un opportuno modulo del kernel, di replicare uno o più device a blocchi in un nodo gemello (solo uno dei due block device può essere montato contemporaneamente in lettura / scrittura). Tra le varie funzionalità è possibile avere una modalità operativa che permette di replicare atomicamente ogni singolo byte senza pertanto causare perdite di dati dovute a crash o spegnimenti improvvisi del sistema.

Dopo una prima fase di installazione dove si è utilizzato il consueto comando:

```
emerge drbd
```

si è provveduto a configurare gli indirizzi IP dei nodi coinvolti nella sincronizzazione, il nome della partizione da replicare, la larghezza di banda massima da utilizzare durante la sincronizzazione dei dati ed il tipo di protocollo utilizzato per la replica dei dati.

³⁹ ENBD: l'Enhanced Network Block Device (<http://www.it.uc3m.es/~ptb/nbd/>) è un meccanismo per esportare dei device a blocchi attraverso delle comuni connessioni di rete.

⁴⁰ MD: il Multidisk Driver permette a GNU/Linux di creare dei raid software (livelli da 0 a 6)

⁴¹ DRBD: <http://www.drbd.org>

⁴² Philipp Reisner: philipp.reisner@linnbit.com

Nel nostro caso è sembrato opportuno utilizzare il protocollo di tipo C che garantisce una perfetta integrità dei dati anche in caso di spegnimento improvviso del sistema e che, soprattutto, garantisce l'atomicità delle transazioni su disco.

Il programma viene distribuito con a corredo un set di utility atto a risincronizzare i dati dopo eventuali guasti hardware ed atto a modificare eventuali parametri operativi del Distributed Replicated Block Device.

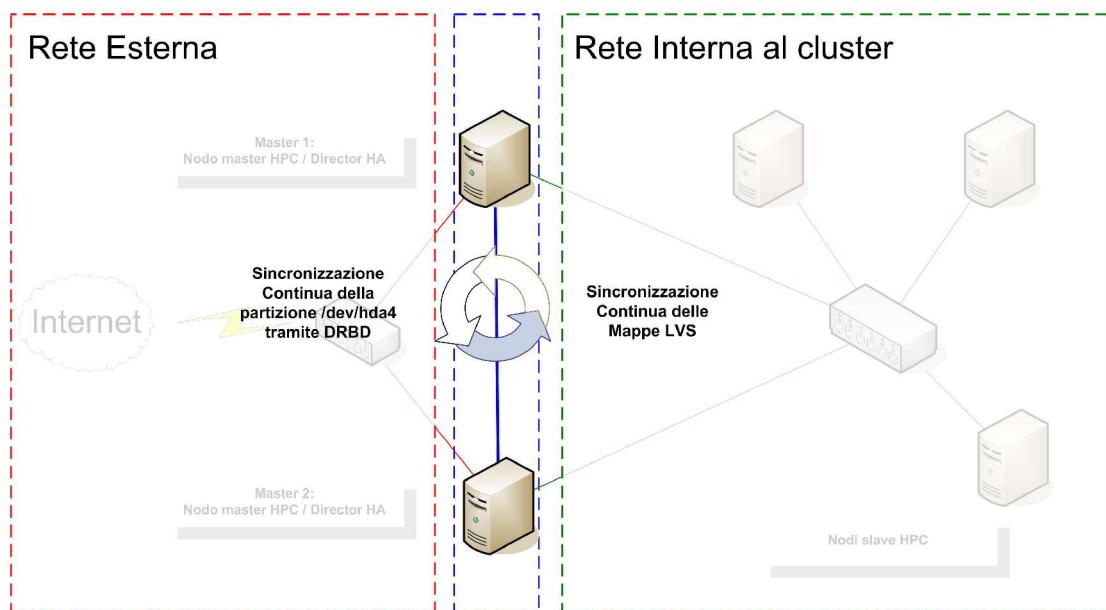


Figura 4.3: Sincronizzazione Realtime della partizione /dev/hda4

4.16 Installazione e configurazione di DRBD ed integrazione con heartbeat

Una volta completati i test che hanno confermato l'affidabilità dell'accoppiata DRBD + Files System “normale”, si è provveduto ad integrare la gestione del Distributed

Replicated Block Device con il demone demandato alla gestione delle funzionalità di clustering HA.

Come prima cosa, si è aggiunto al file di definizione delle risorse */etc/ha.d/haresources* l'indicazione del nuovo DRBD:

```
master1          10.0.3.100          datadisk::drbd0
Filesystem::/dev/nbd/0::/mnt/raid::reiserfs      in.tftpd
nfsserver
```

pertanto si è data la possibilità al demone di heartbeat di montare automaticamente in lettura e scrittura il device esportato via DRBD e si è creato quindi un ambiente nel quale il filesystem replicato via rete è legato in modo indissolubile ai servizi ad alta disponibilità.

4.17 Adattamento del middle-layer alle nuove funzionalità di HA

L'integrazione delle funzionalità ad alta disponibilità nella struttura che permette il boot via rete dei nodi slave HPC ha messo a disposizione nuove potenzialità, che hanno richiesto alcune piccole modifiche al codice componente il middle layer per l'autoconfigurazione dei nodi slave HPC.

In particolar modo, dato che le componenti LVS ed HA installate sui due nodi master HPC / director HA permettono di virtualizzare gli indirizzi IP dei servizi ad alta disponibilità, si è scelto di legare tutte le risorse dei nodi diskless slave HPC non più all'indirizzo di rete reale del nodo master HPC, bensì ad un indirizzo virtuale gestito dal servizio heartbeat.

Utilizzando tale accortezza, si è fatto in modo che i singoli nodi slave HPC non percepiscano eventuali interruzioni di servizio sul nodo master HPC al quale sono

temporaneamente legati. Pertanto, in caso di rottura, avverrà una transizione trasparente delle risorse e si eviteranno fenomeni di stallo e/o disconnessioni improvvise dei servizi.

4.18 Attivazione delle funzionalita' di load balancing: round robin e weighted round robin

Altra caratteristica interessante messa a disposizione delle componenti LVS installate è la possibilità di redirezionare il traffico in ingresso su un indirizzo IP virtuale su un gruppo di nodi, utilizzando una politica round-robin oppure una politica weighted-round-robin.

Tale possibilità ha suggerito di attivare un ulteriore indirizzo IP virtuale sull'interfaccia di rete collegata alla rete pubblica e di redirezionare il traffico SSH in ingresso verso il nodo slave HPC più scarico.

A tal scopo, si è deciso di attivare un IP virtuale, associargli uno schedulatore weighted round robin, installare una coppia di demoni in grado di valutare il carico dei nodi slave HPC e modificare le relative mappe LVS sui nodi master HPC / director HA.

Pertanto, si è provveduto ad installare sui nodi master HPC / director HA attraverso il comando:

```
emerge feedbackd-master
```

un demone capace di ricevere le informazioni di carico dai nodi slave HPC e di modificare le mappe LVS con valori variabili da 0 a 100 indicanti la priorità da attribuire nella scelta del nodo sul quale reindirizzare la richiesta.

Inoltre si è provveduto attraverso il comando:

```
emerge feedbackd-agent
```

ad installare un agente intelligente capace di leggere i valori di carico sulla CPU del singolo nodo slave HPC ed inviare l'informazione al demone *feedbackd-master* capace di rielaborarla e modificare conseguentemente i pesi delle mappe LVS.

L'utilizzo di tale soluzione, seppur semplice, ha permesso di avere un'utilizzabilità ottimale dell'intero sistema distribuito che, pertanto, si trova, in ogni situazione, a lavorare in condizioni ottimali di carico e bilanciamento delle richieste. Inoltre, si ha la possibilità, nel caso in futuro si presentino ulteriori esigenze, di migliorare il sistema di valutazione del carico del singolo nodo in modo da tener conto di eventuali parametri aggiuntivi e di migliorare ulteriormente le performance globali.

CAPITOLO 5

TEST

Dal momento che il cluster realizzato rappresenta la fusione di due tipologie di clustering distinte, ovvero il cluster HPC ed il clustering HA, si è deciso di testare alcuni aspetti significativi di ogni sottoinsieme di funzionalità.

Pertanto si procederà con:

- l'analisi delle performance derivanti dalla compilazione distribuita;
- l'analisi delle performance del Distributed Network Block Device, paragonato ad un file-system locale;
- l'analisi dei tempi medi di transizione delle risorse da un director attivo ad uno operante in modalità passiva in caso di rottura del primo;

5.1 Compilazione distribuita, analisi delle performance

L'introduzione del servizio di compilazione distribuita basato su DISTCC e della cache CCACHE hanno reso possibile effettuare un tuning del sistema veramente meticoloso sui parametri di compilazione utilizzati per compilare i componenti chiave del sistema (Glibc, Berkeley DB, ecc).

Da un'analisi macroscopica è emerso, come possiamo osservare nella tabella e nel grafico seguente, che una tale soluzione porta ad un sostanziale incremento delle performance in fase di compilazione.

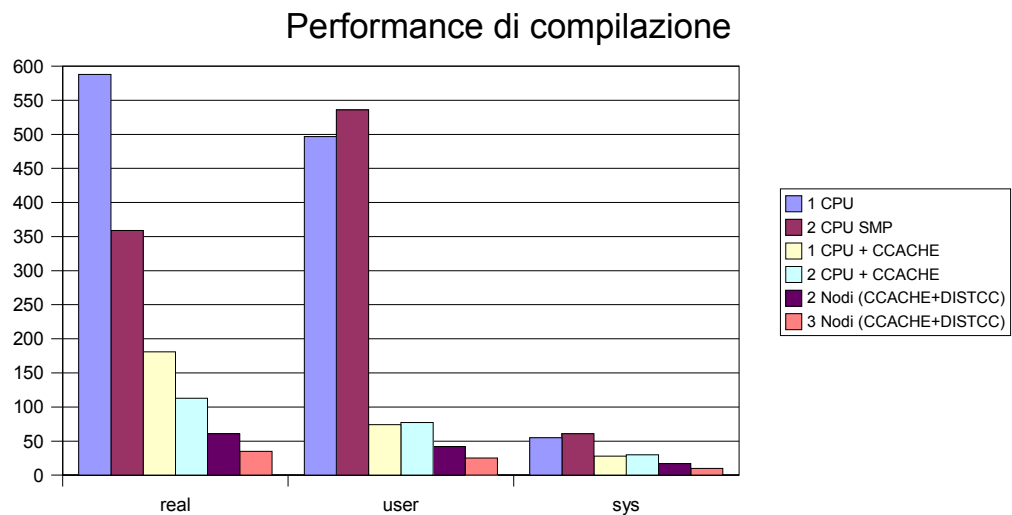
Analisi comparativa dell'incremento di performance dopo l'aggiunta di CCACHE (singolo nodo)

	<i>1 CPU</i>	<i>2 CPU SMP</i>	<i>1 CPU + CCACHE</i>	<i>2 CPU SMP + CCACHE</i>
real	9m48.246s	5m59.021s	3m1.057s	1m53.045s
user	8m17.764s	8m56.643s	1m14.606s	1m17.926s
sys	0m55.512s	1m1.670s	0m28.819s	0m30.997s

CCACHE + DISTCC

	<i>2 Nodi</i>	<i>3 Nodi</i>
real	0m61.081s	0m35.821s
user	0m42.777s	0m25.453s
sys	0m17.142s	0m10.380s

Tabella 5.1: Analisi comparativa incremento performance dopo l'aggiunta di CCACHE (singolo nodo)



5.2 Analisi delle performance di DRBD paragonato ad un block device locale

Per meglio evidenziare i valori di performance derivanti dall'adozione di un block device replicato via rete, quale è DRBD, è necessario analizzare l'influenza che la larghezza di banda della connessione di rete ha sulle performance dell'intero file-system.

Pertanto, utilizzando il tool denominato Bonnie++⁴³, si sono eseguiti alcuni test prestazionali:

- performance del file-system reiserfs su device a blocchi locale
- performance di reiserfs su DRBD (protocollo C⁴⁴) con connessione di rete a 100Mbit/s
- performance di reiserfs su DRBD (protocollo C) con connessione di rete a 1 Gbit/s
- performance di reiserfs su DRBD (protocollo B⁴⁵) con connessione di rete a 1 Gbit/s

Performance su file-system locale:

	Sequential Output						Sequential Input				Random	
Size M	Per Chr K/sec	%CPU	Block K/sec	%CPU	Rewrite K/sec	% CPU	Per Chr K/sec	% CPU	Block K/sec	%CPU	Seeks /sec	%CPU
2016	10767	61	10831	10	4803	5	10699	67	11312	3	244	1
	Sequential Create						Random Create					
Files	Create /sec	% CPU	Read /sec	% CPU	Delete /sec	% CPU	Create /sec	% CPU	Read /sec	% CPU	Delete /sec	% CPU
16	799	7	2509	9	921	7	795	7	2653	9	897	7

Tabella 5.2: Performance del file-system locale

⁴³Bonnie++ è reperibile presso <http://www.coker.com.au/bonnie++/>

⁴⁴Il protocollo C ci garantisce l'atomicità delle transazioni dal momento che invia l'ack unicamente quando il dato è stato scritto anche sull'hard disk remoto.

⁴⁵Il protocollo B non garantisce l'atomicità dal momento che viene inviato un ack nell'istante in cui il dato arriva alla cache dell'host remoto.

Performance del file-system esportato tramite DRBD, protocollo C e connessione operante a 100Mbit/s:

	Sequential Output						Sequential Input				Random	
Size M	Per Chr K/sec	%CPU	Block K/sec	%CPU	Rewrite K/sec	% CPU	Per Chr K/sec	% CPU	Block K/sec	%CPU	Seeks /sec	%CPU
2016	5931	33	5427	4	3727	4	10584	65	11282	3	239,2	1
	Sequential Create						Random Create					
Files	Create /sec	% CPU	Read /sec	% CPU	Delete /sec	% CPU	Create /sec	% CPU	Read /sec	% CPU	Delete /sec	% CPU
16	745	6	2442	8	845	7	762	8	2534	8	836	7

Tabella 5.3: Performance del file-system su DRBD, protocollo C, con ethernet 100 Mbit/s

Performance del file-system esportato tramite DRBD, protocollo C e connessione operante a 1Gbit/s:

	Sequential Output						Sequential Input				Random	
Size M	Per Chr K/sec	%CPU	Block K/sec	%CPU	Rewrite K/sec	% CPU	Per Chr K/sec	% CPU	Block K/sec	%CPU	Seeks /sec	%CPU
2016	7610	23	7629	1	1843	79	12519	59	13825	2	210,2	0
	Sequential Create						Random Create					
Files	Create /sec	% CPU	Read /sec	% CPU	Delete /sec	% CPU	Create /sec	% CPU	Read /sec	% CPU	Delete /sec	% CPU
16	776	9	557	90	817	8	778	9	2952	7	861	8

Tabella 5.4: Performance del file-system su DRBD, protocollo C, con ethernet 1000 Mbit/s

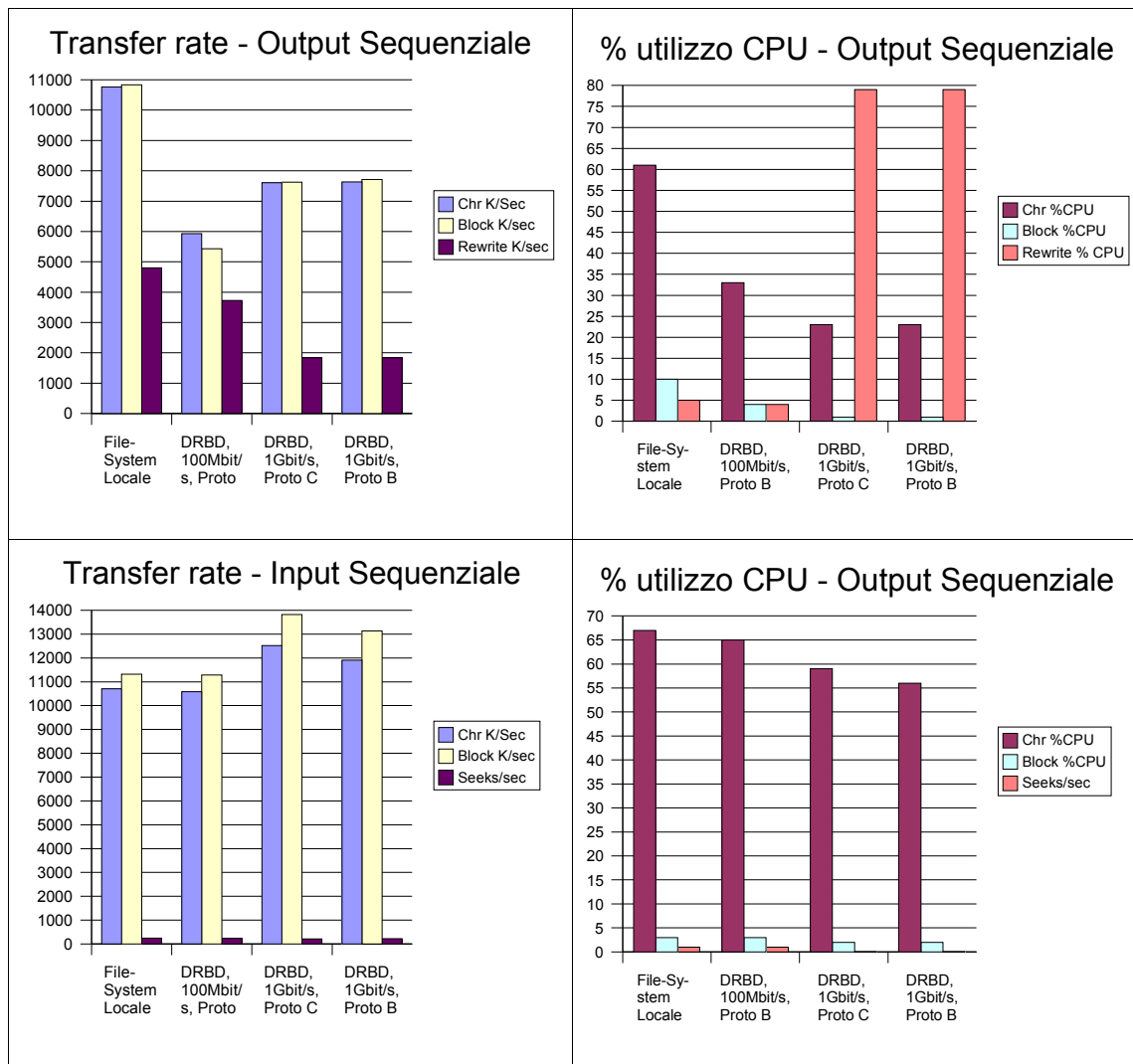
Performance del file-system esportato tramite DRBD, protocollo B e connessione operante a 1Gbit/s:

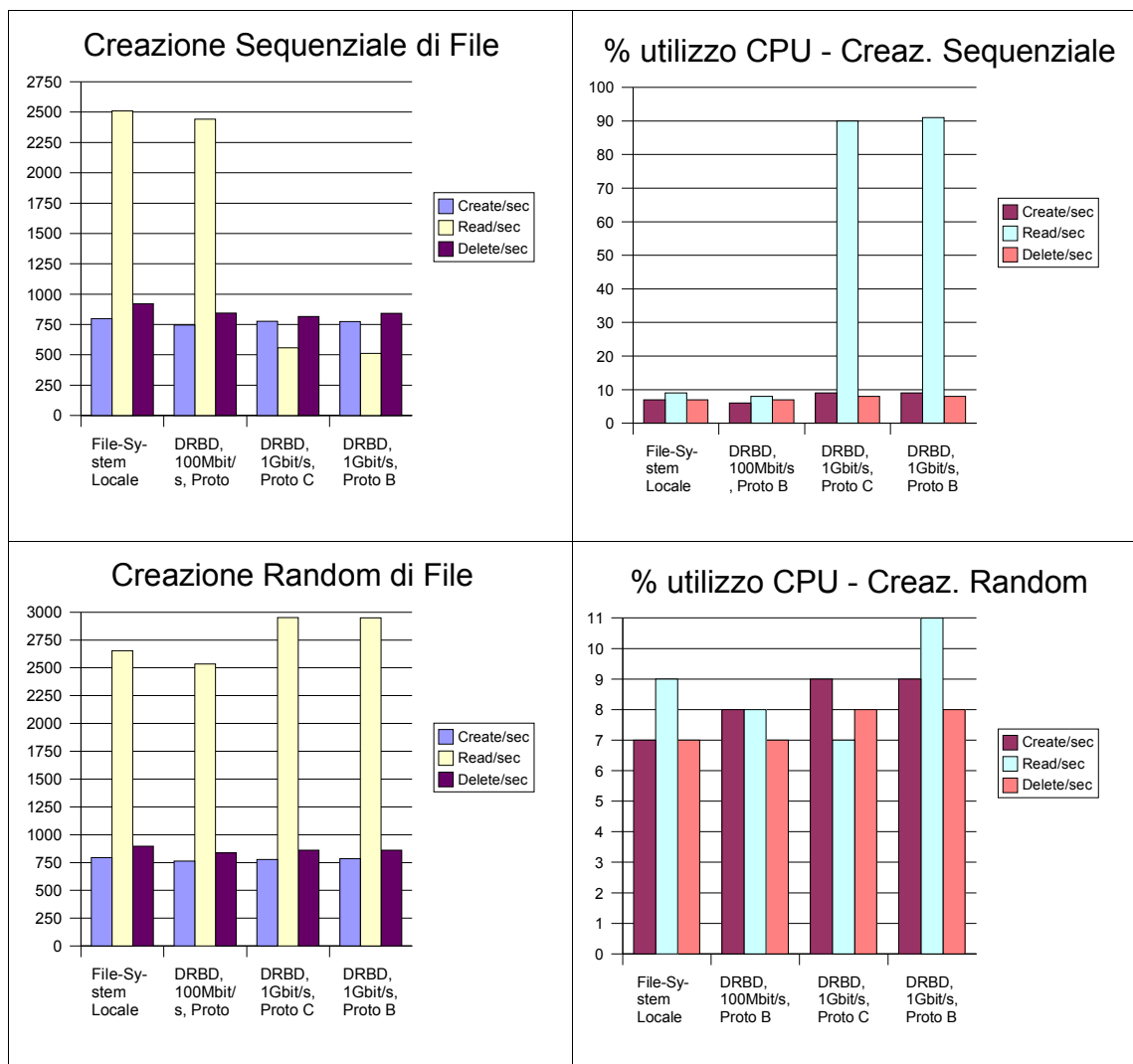
	Sequential Output						Sequential Input				Random	
Size M	Per Chr K/sec	%CPU	Block K/sec	%CPU	Rewrite K/sec	% CPU	Per Chr K/sec	% CPU	Block K/sec	%CPU	Seeks /sec	%CPU
2016	7634	23	7712	1	1839	79	11900	56	13131	2	219	0

	Sequential Create						Random Create					
Files	Create /sec	% CPU	Read /sec	% CPU	Delete /sec	% CPU	Create /sec	% CPU	Read /sec	% CPU	Delete /sec	% CPU
16	774	9	512	91	842	8	785	9	2949	11	860	8

Tabella 5.5: Performance del file-system su DRBD, protocollo B, con ethernet 1000 Mbit/s

Come risulta evidente dai seguenti grafici, la larghezza di banda sulla connessione utilizzata per scambiare i dati DRBD influenza in buona misura le performance del file-system utilizzato.





5.3 Analisi dei tempi medi di transizione delle risorse da un director attivo ad uno operante in modalità passiva

Dai test condotti in varie condizioni di utilizzo del cluster è emerso che il tempo di commutazione delle risorse da un master attivo ad uno operante in modalità passiva è assolutamente indipendente da ogni fattore di carico, mentre risulta essere influenzato dai i settaggi del demone heartbeat, in particolare dal numero di tentativi di connessione

e dal numero di controlli che un director operante in modalità passiva deve fare prima di dichiarare il partner come defunto ed iniziare la transizione delle risorse.

Tipicamente, si sono rilevati dei tempi di commutazione variabili da 6 a 15 secondi.

CONCLUSIONI

Grazie al lavoro di tesi svolto, si è potuto verificare come sia possibile realizzare un sistema distribuito dotato di funzionalità per il calcolo ad alte performance ed al contempo dotato di funzionalità per l'erogazione ad alta disponibilità del servizio, utilizzando unicamente il sistema operativo GNU/Linux ed il software open source e pertanto annullando completamente il costo delle licenze.

Tuttavia, durante la realizzazione delle porzione ad alta disponibilità, è emerso come alcuni software distribuiti sotto licenze open source siano al momento non adatti ad ambienti di produzione e pertanto non utilizzabili in un contesto critico come potrebbe essere un ospedale e nella fattispecie un laboratorio per gli esami mammografici.

D'altro canto la ricchezza di alternative in termini di software utilizzabili ci ha permesso di ripiegare su implementazioni che, pur essendo meno flessibili rispetto ad un file system distribuito/clusterizzato, sono in grado di fornire performance e stabilità sufficienti ad un utilizzo in un ambiente di produzione.

Evoluzioni future del presente lavoro di tesi potrebbero portare all'integrazione di nuove release di software quali il Global File System di RedHAT, che una volta stabilizzato ed integrato appieno nei nuovi kernel della serie 2.6.x, potrebbe incrementare notevolmente le performance dell'intero cluster. L'adozione di un tale strumento porterebbe ad un maggiore linearità nella gestione dei file all'interno di ogni nodo del cluster e ci permetterebbe di eliminare sia DRBD, sia NFS senza perdere in funzionalità e performance.

BIBLIOGRAFIA

T. Cecchi, “*GFS Global File system*”, CINECA.

Cluster NFS homepage, <http://clusternfs.sourceforge.net>

R. Droms, “*Dynamic Host Configuration Protocol*”, RFC 2131 – Network Working Group, Prentice Hall International edition.

L. Ferreira, G. Kettmann, A. Thomasch, E. Silcocks, J. Chen, J.C. Daunois, J. Ihamo, M. Harada, S. Hill, W. Bernocchi, E. Ford, “*Linux HPC Cluster Installation*”, IBM Redbooks, <http://www.ibm.com/redbooks/>

“*FreeBSD Handbook*”, <http://www.freebsd.org>

G. Gousios, “*Root over NFS – Another Approach*”, University of Aegean, Greece.

W. Gropp, E. Lusk, “*Installation guide and user's guide to MPICH a Portable implementation of MPI*”, <http://www-unix.mcs.anl.gov/mpi/mpich>

IBM @server White Paper, Norm Snyder, “*IBM Linux Cluster*”, <http://www.ibm.com>

“*Linux HA Wiki*”, <http://wiki.linux-ha.org/>

L. Marowsky-Brée, “*A new cluster resource manager for heartbeat*”, UKUUG LISA/Winter Conference High-Availability and Reliability, Bournemouth, UK, 25-26 February, 2004.

L. Marowsky-Brée, “*Deploying heartbeat*”, UKUUG LISA/Winter Conference High-Availability and Reliability, Bournemouth, UK, 25-26 February, 2004.

P. Mastroserio, F.M. Taurino, G. Tortone, “*Mosix: High performance Linux farm*”, INFN Italy.

J. McQuillan, “*LTSP – Linux Terminal Server Project – v 3.0*”, <http://www.ltsp.org/>

Moshe Bar, S. Cozzini, M. Davini, A. Marmodoro “*openMosix vs Beowulf: a case study*”.

Moshe Bar, INFN Democritos, openMosix Project, Department of physics University of Pisa, “*Linux file system*”, Mc-Graw Hill.

M. Pool, “*distcc, a fast free distributed compiler*”, linux.conf.au, Adelaide, 2004.

D. Robbins, “*Linux clustering with Mosix*”, <http://www.ibm.com/>

A.L. Robertson, “*The Evolution of the Linux-HA Project*”, UKUUG LISA/Winter Conference High-Availability and Reliability, Bournemouth, UK, 25-26 February, 2004.

A.L. Robertson, “*PILS: A generalized Plugin and Interface Loading System*”, *Proceedings of the 4th Linux Symposium, Ottawa, Canada, June 26-29, 2002.*

A.L. Robertson, “*Linux-HA Heartbeat Design*”, Proceedings of the 4th International Linux Showcase and Conference, Atlanta, October 10-14 2000.

A.L. Robertson, "*Programming for Linux-HA*", UKUUG LISA/Winter Conference High-Availability and Reliability, Bournemouth, UK, 25-26 February, 2004.

B.A. Shirazi, A.R. Hurson, K.M. Kavi, "*Scheduling and balancing in parallel and distributed system*", IEEE Computer Society Press.

Silberschatz, Galvin, Gagne, "*Applied operating system concepts*", John Wiley & sons, INC.

H. Stern, "*Managing NFS and NIS*", O'Reilly Associated INC.

A. Tanenbaum, "*Computer Networks*", Prentice Hall International edition.

A. Tanenbaum, "*Modern operating system*".

A. Tanenbaum, "*Distributed operating system*", Prentice Hall International edition

APPENDICE

File: /etc/fstab (nodi master)

```
/dev/hda2          /                  ext3              noatime
0 0
/dev/hda1          none              swap              sw
0 0
/dev/cdroms/cdrom0  /mnt/cdrom        iso9660           noauto,ro
0 0
/dev/fd0           /mnt/floppy       auto              noauto
0 0
/dev/nbd/0         /diskless/        reiserfs
notail,noatime,noauto 0 0
none              /proc             proc              defaults
0 0
none              /dev/shm          tmpfs             defaults
0 0
```

File: /etc/openldap/slapd.conf (nodi master)

```
include            /etc/openldap/schema/core.schema
include            /etc/openldap/schema/cosine.schema
include            /etc/openldap/schema/inetorgperson.schema
include            /etc/openldap/schema/nis.schema
include            /etc/openldap/schema/samba.schema
include            /etc/openldap/schema/qmail.schema
include            /etc/openldap/schema/krb5-kdc.schema
include            /etc/openldap/schema/kerberosobject.schema

pidfile            /var/run/openldap/slapd.pid
argsfile           /var/run/openldap/slapd.args

allow bind_v2
loglevel 0

repllogfile /var/lib/openldap-slurp/replication.log

#####
#
# ldbm database definitions
#####
#

database           ldbm
suffix             "dc=fn,dc=csr,dc=unibo,dc=it"
lastmod            on
rootdn             "cn=Manager,dc=fn,dc=csr,dc=unibo,dc=it"

replica host=10.0.3.2:389
               binddn="cn=replica,dc=fn,dc=csr,dc=unibo,dc=it"
```

```

        bindmethod=simple credentials=linux

rootpw          {SHA}SuPeTWjeDXeN0lQeanX0VdMGU+M=
directory       /var/lib/ldap-data

index    objectClass    eq

access to attrs=gecos,description,loginShell
        by dn="uid=proxyuser,ou=People,dc=fn,dc=csr,dc=unibo,dc=it"
read
        by dn="cn=replica,dc=fn,dc=csr,dc=unibo,dc=it" read
        by self write

access to dn=".*,ou=People,dc=fn,dc=csr,dc=unibo,dc=it"
attr="userPassword"
        by dn="uid=root,ou=People,dc=fn,dc=csr,dc=unibo,dc=it" write
        by dn="uid=proxyuser,ou=People,dc=fn,dc=csr,dc=unibo,dc=it"
read
        by dn="cn=replica,dc=fn,dc=csr,dc=unibo,dc=it" read
        by anonymous auth
        by self write
        by * search

ccess to dn=".*,ou=Group,dc=fn,dc=csr,dc=unibo,dc=it"
        by dn="uid=root,ou=People,dc=fn,dc=csr,dc=unibo,dc=it" write
        by dn="uid=proxyuser,ou=People,dc=fn,dc=csr,dc=unibo,dc=it"
read
        by dn="cn=replica,dc=fn,dc=csr,dc=unibo,dc=it" read

access to *
        by dn="uid=root,ou=People,dc=fn,dc=csr,dc=unibo,dc=it" write
        by dn="cn=replica,dc=fn,dc=csr,dc=unibo,dc=it" read
        by dn="uid=proxyuser,ou=People,dc=fn,dc=csr,dc=unibo,dc=it"
read
        by users read
        by anonymous auth

```

File: /etc/dhcp/dhcpd.conf (nodi master)

```
# DHCP configuration file for DHCP ISC 3.0

ddns-update-style none;

# Definition of PXE-specific options
# Code 1: Multicast IP address of boot file server
# Code 2: UDP port that client should monitor for MTFTP responses
# Code 3: UDP port that MTFTP servers are using to listen for MTFTP
requests
# Code 4: Number of seconds a client must listen for activity before
trying
#           to start a new MTFTP transfer
# Code 5: Number of seconds a client must listen before trying to
restart
#           a MTFTP transfer

option space PXE;
option PXE.mtftp-ip                code 1 = ip-address;
option PXE.mtftp-cport             code 2 = unsigned integer 16;
option PXE.mtftp-sport             code 3 = unsigned integer 16;
option PXE.mtftp-tmout             code 4 = unsigned integer 8;
option PXE.mtftp-delay             code 5 = unsigned integer 8;
option PXE.discovery-control       code 6 = unsigned integer 8;
option PXE.discovery-mcast-addr    code 7 = ip-address;

subnet 10.0.3.0 netmask 255.255.255.0 {
    server-name                    "master";
    # Use your gateway IP, if required
    option routers                 10.0.3.100;
    # Use your DNS IP, if required
    option domain-name-servers     10.0.3.1,10.0.3.2;
    option domain-name             "fn.csr.unibo.it";

    class "pxeclients" {
        match if substring (option vendor-class-identifier, 0, 9)
= "PXEClient";
        option vendor-class-identifier "PXEClient";
        vendor-option-space PXE;

        # At least one of the vendor-specific PXE options must be
set in
        # order for the client boot ROMs to realize that we are a
PXE-compliant
        # server. We set the MCAST IP address to 0.0.0.0 to tell
the boot ROM
        # that we can't provide multicast TFTP (address 0.0.0.0
means no
        # address).
```

```

        option PXE.mtftp-ip 0.0.0.0;

        # This is the name of the file the boot ROMs should
download. filename "pxelinux.0";
        # This is the name of the server they should get it from.
        # Use the master's IP
        next-server 10.0.3.100;
    }

    #pool {
        max-lease-time 86400;
        default-lease-time 86400;
        # This prevents unlisted machines from getting an IP
        #deny unknown clients;
    }

    host slave1 {
        # Use your slave's MAC address
        hardware ethernet          00:04:75:D6:1E:8D;
        # Give your slave a static IP
        fixed-address               10.0.3.11;
        # Use your slave hostname
        option host-name           "slave1";
    }

    host slave2 {
        # Use your slave's MAC address
        hardware ethernet          00:01:02:1e:c9:5f;
        # Give your slave a static IP
        fixed-address               10.0.3.12;
        # Use your slave hostname
        option host-name           "slave2";
    }

    host slave3 {
        # Use your slave's MAC address
        hardware ethernet          00:04:75:d6:1c:fb;
        # Give your slave a static IP
        fixed-address               10.0.3.13;
        # Use your slave hostname
        option host-name           "slave3";
    }
}

```


File: /diskless/pxelinux.cfg/default (nodi master)

```
DEFAULT /bzImage
APPEND ip=dhcp root=/dev/ram0 initrd=/initrd init=/linuxrc
```

heartbeat

File: /etc/ha.d/ha.cf (nodi master)

```
logfile          /var/log/ha-log
keepalive 1
deadtime 10
initdead 20
auto_failback on
mcast eth0 224.0.0.0 694 1 1
baud 19200
serial /dev/ttyS0
node master1 master2
realtime on
watchdog /dev/watchdog
```

File: /etc/ha.d/haresources (nodi master)

```
master1 10.0.3.100 datadisk::drbd0
Filesystem::/dev/nbd/0::/mnt/raid::reiserfs dhcp in.tftpd nfsserver

master1 137.204.72.198/24/eth1 route lvs nat
```

File: /etc/ha.d/authkeys (nodi master)

```
auth 1
1 crc
```

File: /etc/ha.d/resource.d/nfsserver (nodi master)

```
#!/bin/sh

unset LC_ALL;
export LC_ALL
unset LANGUAGE;
export LANGUAGE
prefix=/usr
exec_prefix=/usr

. /etc/ha.d/shellfuncs
```

```

# The binary locations
exportfs=/usr/sbin/exportfs
statd=/sbin/rpc.statd
rquotad=/usr/sbin/rpc.rquotad
nfsd=/usr/sbin/rpc.nfsd
mountd=/usr/sbin/rpc.mountd
nfsd_nproc=16

start_statd () {
    # Don't start rpc.statd if already started by init.d/nfsmount
    killall -0 rpc.statd &>/dev/null && return 0
    $statd -n cluster 1>&2
}

stop_statd () {
    # Make sure it's actually running
    killall -0 rpc.statd &>/dev/null || return 0
    # Okay, all tests passed, stop rpc.statd
    killproc $statd
}

case "$1" in
    'start')
        start_statd

        # Exportfs likes to hang if networking isn't working.
        # If that's the case, then try to kill it so the
        # bootup process can continue.
        if grep -q '^/' /etc/exports &>/dev/null; then
            $exportfs -r 1>&2 &
            pid=$!
            ( sleep 30; kill -9 $pid &>/dev/null ) &
            wait $pid
        fi

        #if [ -x $rquotad ]; then
            #ebegin "Starting NFS rquotad"
            #$rquotad
        #fi

        $nfsd $nfsd_nproc

        # Check if we support NFSv3
        rpcinfo -u localhost nfs 3 &>/dev/null || \
            RPCMOUNTDOPTS="$RPCMOUNTDOPTS --no-nfs-version
3"

        $mountd $RPCMOUNTDOPTS 1>&2

```

```

;;

'pre-start')
;;

'post-start')
;;

'stop')
    killproc $mountd
    killproc $nfsd
    #killproc $rquotad
    stop_statd
    killproc $nfsd
    #killall -INT nfsd
    #killall -9 nfsd
    killall -9 lockd
    sleep 1s
    killall -9 nfsd
;;

'pre-stop')
;;
'post-stop')
;;
*)
    echo "Usage: $0 { start | pre-start | post-start | stop |
pre-stop | post-stop }"
;;
esac

exit 0

```

File: /etc/ha.d/resource.d/dhcp (node master)

```

#!/bin/bash

IFACE=eth2

start() {
    if [ ! -f "/var/lib/dhcp/dhcpd.leases" ] ; then
        touch "${CHROOT}/var/lib/dhcp/dhcpd.leases" || return
1
    fi

    chown dhcp:dhcp "/var/lib/dhcp/dhcpd.leases" || return 1

    start-stop-daemon --start --quiet --exec /usr/sbin/dhcpd \
        -- -pf /var/run/dhcp/dhcpd.pid -q \
        -user dhcp -group dhcp ${DHCPD_OPTS} \
        ${IFACE}
}

```

```

stop() {
    killproc /usr/sbin/dhcpd && rm -f /var/run/dhcp/dhcpd.pid
}

case "$1" in
    'start')
        start
    ;;
    'pre-start')
    ;;
    'post-start')
    ;;
    'stop')
        stop
    ;;
    'pre-stop')
    ;;
    'post-stop')
    ;;
    *)
        echo "Usage: $0 { start | pre-start | post-start | stop |
pre-stop | post-stop }"
    ;;
esac

exit 0

```

File: /etc/ha.d/resource.d/in.tftpd (nodi master)

```

#!/bin/bash

INTFTPD_PATH="/diskless"
INTFTPD_OPTS="-l -v -s ${INTFTPD_PATH}"

start() {
    /usr/sbin/in.tftpd ${INTFTPD_OPTS}
}

stop() {
    killproc /usr/sbin/in.tftpd
}

case "$1" in
    'start')
        start

```

```

;;

'pre-start')
;;

'post-start')
;;

'stop')
        stop
;;

'pre-stop')
;;
'post-stop')
;;
*)
        echo "Usage: $0 { start | pre-start | post-start | stop |
pre-stop | post-stop }"
;;
esac

exit 0

```

File: /etc/ha.d/resources.d/lvs (nodi master)

```

#!/bin/sh

MASTERS="10.0.3.1 10.0.3.2"
NODES="10.0.3.11 10.0.3.12 10.0.3.13"
VIP_EXT="137.204.72.198"
VIP_EXT_BROADCAST="137.204.72.255"
VIP_EXT_NETMASK="255.255.255.0"
VIP_EXT_ALIAS="198"
VIP_EXT_IFACE="eth1"

VIP_INT="10.0.3.100"
VIP_INT_IFACE="eth2"

IPVSADM=/sbin/ipvsadm

case "$1" in

    'start')

        # set ip_forward ON for vs-nat director (1 on, 0 off).
        echo "1" >/proc/sys/net/ipv4/ip_forward

        # director is gw for realservers
        # turn OFF icmp redirects (1 on, 0 off)
        echo "0" >/proc/sys/net/ipv4/conf/all/send_redirects
    ;;

```

```

>/proc/sys/net/ipv4/conf/default/send_redirects          echo    "0"
>/proc/sys/net/ipv4/conf/$VIP_EXT_IFACE/send_redirects    echo    "0"

# LVS Services SSH
# install LVS services with ipvsadm
$IPVSADM -A -t $VIP_EXT:22 -s wrr

for i in $NODES; do
    $IPVSADM -a -t $VIP_EXT:22 -r $i:22 -w 100 -m
done

# DNS
$IPVSADM -A -t $VIP_INT:53 -s wrr
$IPVSADM -A -u $VIP_INT:53 -s wrr

for i in $MASTERS; do
    $IPVSADM -a -t $VIP_INT:53 -r $i:53 -w 100 -m
    $IPVSADM -a -u $VIP_INT:53 -r $i:53 -w 100 -m
done

feedbackd-master

;;
'stop')
    if [ -x $IPVSADM ]; then
        #$IPVSADM -C
        echo ""
    fi
    killall -INT feedbackd-master
;;
*)
    echo  "Usage: $0 { start | stop }"
;;
esac

exit 0

```

File: /etc/ha.d/resources.d/nat (nodi master)

```

#!/bin/sh
IPTABLES=/sbin/iptables
EXT_NET=eth1

start() {
    #$IPTABLES -F OUTPUT; $IPTABLES -F INPUT ; $IPTABLES -F
    FORWARD

    # flush any old rules
    #$IPTABLES -F -t nat

```

```

        # turn on NAT (IP masquerading for outgoing packets)
        $IPTABLES -t nat -A POSTROUTING -o $EXT_NET -j SNAT --to
137.204.72.198

        # enable IP forwarding (of incoming packets)
        echo 1 > /proc/sys/net/ipv4/ip_forward

        $IPTABLES -t nat -A PREROUTING -p tcp --dport 2020 -i $EXT_NET
-j DNAT --to 10.0.3.1:22
        $IPTABLES -t nat -A PREROUTING -p tcp --dport 2021 -i $EXT_NET
-j DNAT --to 10.0.3.2:22
        $IPTABLES -t nat -A PREROUTING -p tcp --dport 2022 -i $EXT_NET
-j DNAT --to 10.0.3.11:22
        $IPTABLES -t nat -A PREROUTING -p tcp --dport 2023 -i $EXT_NET
-j DNAT --to 10.0.3.12:22
        $IPTABLES -t nat -A PREROUTING -p tcp --dport 2024 -i $EXT_NET
-j DNAT --to 10.0.3.13:22
        $IPTABLES -t nat -A PREROUTING -p tcp --dport 2025 -i $EXT_NET
-j DNAT --to 10.0.3.14:22

        $IPTABLES -t nat -A PREROUTING -p tcp --dport 81 -i $EXT_NET
-j DNAT --to 10.0.3.2:80
    }

    stop() {
        $IPTABLES -F OUTPUT; $IPTABLES -F INPUT ; $IPTABLES -F FORWARD
        $IPTABLES -F -t nat
    }

    case "$1" in
        'start')
            start
        ;;
        'stop')
            stop
        ;;
        *)
            echo "Usage: $0 { start | stop }"
        ;;
    esac

```

Desidero ringraziare:

Linus Torvalds che con Linux ha cambiato la mia vita di informatico.

La comunità degli sviluppatori di Gentoo Linux che mi ha accolto nella loro grande “famiglia” permettendomi di integrare le mie ricerche nella loro stupenda distribuzione.

Il professor Campanini che mi ha permesso di fare questa bellissima esperienza di tesi.

Il dottor Roffilli per la sua cordialità, per la sua disponibilità ed il suo appoggio.

I ragazzi del laboratorio di Biofisica delle Reti Neurali – Radiolab, per i bei momenti passati assieme.