Università degli Studi di Perugia
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

# Uso di macchine virtuali (XEN) per garantire servizi di Grid.

Candidato
**Álvaro López García**

Relatore
*Prof.* **Leonello Servoli**

Correlatore
**Mirko Mariotti**

Anno Accademico 2005-2006

# Indice


**1 Introduzione a Grid e all'alta disponibilità** **3**
1.1 Grid . . . 3
1.1.1 Architettura di Grid . . . 4
1.2 Worldwide LHC Computing Grid (LCG) . . . 5
1.2.1 Il CERN e il LHC . . . 5
1.2.2 Il Worldwide LHC Computing Grid . . . 5
1.2.3 La INFN-Grid . . . 6
1.2.4 Struttura di WLCG . . . 6
1.3 Alta disponibilità . . . 7
1.3.1 Alta disponibilità tramite backups . . . 8
1.3.2 Alta disponibilità tramite ridondanza fisica . . . 8
1.3.3 Alta disponibilità tramite virtualizzazione . . . 9

**2 Virtualizzazione** **11**
2.1 Cos'è la virtualizzazione . . . 11
2.2 Teoria di Popek e Goldberg . . . 11
2.2.1 Insiemi di istruzioni . . . 12
2.2.2 Teoremi . . . 12
2.2.3 Effetti di la teoria di Popek e Goldberg . . . 12
2.3 Tipi di virtualizzazione . . . 13
2.3.1 Emulazione . . . 13
2.3.2 Virtualizzazione . . . 13
2.3.3 Virtualizzazione a livello di SO . . . 14
2.3.4 Paravirtualizzazione . . . 14
2.4 Xen . . . 14
2.4.1 Cos'è Xen . . . 14
2.4.2 Paravirtualizzazione in Xen . . . 15
2.4.3 Architettura di Xen . . . 16
2.4.4 I daemon di Xen . . . 17
2.4.5 Caratteristiche di Xen . . . 17
2.5 Vantaggi della virtualizzazione . . . 18

2.6 Alta disponibilità tramite virtualizzazione . . . . . . . . . . . . . . . 19

**3 Il prototipo: analisi delle possibilità e test delle componenti 21**

3.1 Struttura . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
3.2 Soluzioni distinte per lo storage . . . . . . . . . . . . . . . . . . . . 22
3.2.1 Block device remoti via hardware . . . . . . . . . . . . . . . . 23
3.2.2 Block device remoti via software . . . . . . . . . . . . . . . . . 24
3.2.3 Filesystem distribuiti . . . . . . . . . . . . . . . . . . . . . . 25
3.3 Test di compatibilità . . . . . . . . . . . . . . . . . . . . . . . . . . 26
3.3.1 Test di sistemi *host* . . . . . . . . . . . . . . . . . . . . . . . 26
3.3.2 Test di sistemi *guest* . . . . . . . . . . . . . . . . . . . . . . 26
3.3.3 Test di dispositivi storage . . . . . . . . . . . . . . . . . . . . 26
3.4 Test di I/O . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
3.4.1 IOzone . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
3.4.2 Caratteristiche delle macchine utilizzate nei test . . . . . . . . 27
3.4.3 Interpretazione dei risultati . . . . . . . . . . . . . . . . . . . 28
3.4.4 Risultati . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
3.4.5 Confronto risultati . . . . . . . . . . . . . . . . . . . . . . . . 34
3.5 Healthcheck . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
3.5.1 Heartbeat . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
3.5.2 Nagios . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
3.5.3 Healthcheck propio . . . . . . . . . . . . . . . . . . . . . . . 42

**4 Il prototipo: realizzazione 43**

4.1 Sistema di storage . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
4.2 Healthcheck . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43
4.2.1 Master . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
4.2.2 Slave . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
4.3 Il prototipo . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

**5 Prospettive future 49**

**A Installazione e uso di Xen 53**

A.1 Installazione di Xen . . . . . . . . . . . . . . . . . . . . . . . . . . 53
A.2 Configurazione di Xen . . . . . . . . . . . . . . . . . . . . . . . . . 53
A.2.1 Configurazione di xend . . . . . . . . . . . . . . . . . . . . . . 53
A.2.2 Configurazione dei macchine virtuali . . . . . . . . . . . . . . 54
A.2.3 pyGRUB . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
A.3 Uso di Xen . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
A.3.1 Lanciamento di Xen . . . . . . . . . . . . . . . . . . . . . . . 55
A.3.2 Il tool xm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
A.4 Creazione di una VM SL4 . . . . . . . . . . . . . . . . . . . . . . . 56
A.4.1 Installazione di SL4 . . . . . . . . . . . . . . . . . . . . . . . 56

A.4.2 Compilazione del kernel 2.6.16-xen . . . . . . . . . . . . . . 56
A.4.3 Creazione e prova delle macchine virtuali . . . . . . . . . . . 57

**B Uso di block device via rete 59**
B.1 iSCSI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59
B.1.1 Introduzione . . . . . . . . . . . . . . . . . . . . . . . . . 59
B.1.2 Installazione di un target . . . . . . . . . . . . . . . . . . . 59
B.1.3 Installazione di un initiator . . . . . . . . . . . . . . . . . . 60
B.2 GNBD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
B.2.1 Introduzione . . . . . . . . . . . . . . . . . . . . . . . . . 61
B.2.2 Installazione di GNBD . . . . . . . . . . . . . . . . . . . . 61
B.2.3 Esportazione e Importazione . . . . . . . . . . . . . . . . . 62

**C Healthcheck 63**
C.1 Uso . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
C.2 Sorgenti . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
C.2.1 master.c . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
C.2.2 slave.c . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
C.2.3 control.h . . . . . . . . . . . . . . . . . . . . . . . . . . . 84
C.2.4 File di congurazione . . . . . . . . . . . . . . . . . . . . . 84

# Elenco delle figure


1.1 Livelli di Grid. (Immagine: CERN) . . . . 4
1.2 Il LCG nel mondo. (Immagine: CERN) . . . . 5
1.3 Struttura geografica INFN-Grid. . . . . 7
1.4 Struttura di base di WLCG. . . . . 8
1.5 Schema della ridondanza fisica. . . . . 9

2.1 Confronto del rendimento (Immagine: Xen) . . . . 15
2.2 Livelli di privilegi . . . . 16
2.3 Architettura di Xen . . . . 17
2.4 Struttura di rete con macchine virtuali. . . . . 19
2.5 Ripristino di due servizi virtualizzati . . . . 20

3.1 Prototipo di alta disponibilità tramite Xen. . . . . 22
3.2 Le topologie FC. . . . . 23
3.3 Architettura di iSCSI. . . . . 25
3.4 Schema di filesystem distribuiti. . . . . 25
3.5 Esempio di risultato di un test con IOzone. . . . . 28
3.6 Test di lettura a 64 bit. . . . . 29
3.7 Test di scrittura a 64 bit. . . . . 29
3.8 Test di lettura a 32 bit. . . . . 30
3.9 Test di scrittura a 32 bit. . . . . 30
3.10 Test di lettura a 64 bit. . . . . 31
3.11 Test di scrittura a 64 bit. . . . . 31
3.12 Test di lettura a 32 bit. . . . . 32
3.13 Test di scrittura a 32 bit. . . . . 32
3.14 Test di lettura a 64 bit. . . . . 33
3.15 Test di scrittura a 64 bit. . . . . 33
3.16 Test di lettura a 32 bit. . . . . 34
3.17 Test di scrittura a 32 bit. . . . . 34
3.18 Confronto 2D risultati scrittura a 32 bit. . . . . 36
3.19 Confronto 3D risultati scrittura a 32 bit. . . . . 36
3.20 Confronto 2D risultati lettura a 32 bit. . . . . 37

3.21 Confronto 3D risultati lettura a 32 bit. . . . . . . . . . . . . . . . . . . 37
3.22 Confronto 2D risultati scrittura a 64 bit. . . . . . . . . . . . . . . . . . 38
3.23 Confronto 3D risultati scrittura a 64 bit. . . . . . . . . . . . . . . . . . 38
3.24 Confronto 2D risultati lettura a 64 bit. . . . . . . . . . . . . . . . . . . 39
3.25 Confronto 3D risultati lettura a 64 bit. . . . . . . . . . . . . . . . . . . 39
3.26 Schema di hearbeat. . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
3.27 Screenshot di nagios. . . . . . . . . . . . . . . . . . . . . . . . . . . 41

A.1 Menu di pyGRUB . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

# Introduzione

## Introduzione generale

La ridondanza e la capacità di funzionare anche in caso di guasto (alta disponibilità) devono essere una caratteristica fondamentale di qualunque sistema computazionale orientato a fornire servizi, anche in ambito scientifico –come il caso del Grid Computing–, in quanto non è ammissibile la perdita dei dati e le conseguenze –in quanto a tempo di calcolo e potenza computazionale– che quello implica. Inoltre, ci sono servizi Grid che devono essere disponibili almeno il 99% del tempo per un buon funzionamento della struttura computazionale.

D'altra parte, le differenti e più recenti tecniche di virtualizzazione consentono di isolare la esecuzione di un sistema operativo dalla macchina fisica con un rendimento praticamente similare alla esecuzione senza virtualizzazione. Quindi, l'uso di queste tecnologie di virtualizzazione può apportare grandi vantaggi nel disegno e nell'implementazione di una potente soluzione di alta disponibilità.

## Struttura della tesi

La tesi è basata in 5 capitoli con la seguente struttura:

**Capitolo 1** Breve descrizione di cos'è Grid Computing, come funziona l'INFN Grid e il significato del concetto di alta disponibilità.

**Capitolo 2** in questo capitolo viene descritta la virtualizzazione, il suo funzionamento e le distinte soluzioni disponibili per implementarla.

**Capitolo 3** in questo capitolo viene proposto il prototipo concettuale per realizzare l'alta disponibilità mediante l'uso della virtualizzazione; inoltre vengono presentati i test fatti sulle differenti soluzioni disponibili.

**Capitolo 4** in questo capitolo viene definita ed implementata una prima versione del prototipo proposto nel capitolo precedente.

**Capitolo 5** in questo capitolo vengono descritte le possibili azioni da realizzare per ottenere una implementazione più solida ed efficiente del prototipo di alta disponibilità mediante l'uso di macchine virtuali.

**Appendice A** appendice nella quale viene descritta la installazione e l'uso di Xen 3.0.2 e i problemi atisi durante la installazione. Inoltre, vengono descritte la creazione di distinte immagini minimali funzionanti di Scientific Linux.

**Appendice B** appendice nella quale c'è una spiegazione di come instalare e configurare i distinti dispositivi di storage utilizati nei test fatti nella tesi.

**Appendice C** appendice nella quale viene introdotto una prima versione dil servizio di supervisione del prototipo proposto nel Capitolo 4.

# Capitolo 1

# Introduzione a Grid e all'alta disponibilità

## 1.1 Grid

Il Grid Computing[1] è un paradigma di computazione distribuita (ideato dai fondatori[2] del progetto Globus[3]) nel quale tutte le risorse di un numero indeterminato di computer (sia risorse di calcolo, sia risorse di storage) geograficamente distribuite vengono inglobate –per mezzo di una tecnologia di rete standard– per essere trattate come un unico supercomputer, in maniera trasparente per gli utenti.

Il modello di computazione implementato per Grid ha come obiettivo quello di fornire i mezzi necessari per risolvere problemi computazionali complessi e troppo grandi per un unico supercomputer –come quelli di *Protein Folding*, modellazione climatica, ecc– mentre è anche usabile per problemi di tipo più semplice.

Questo paradigma si fonda su cinque principi:

- **Condivisione di risorse**: L'idea di partenza di Grid è la esistenza e condivisione di differenti risorse –di differenti proprietari, con diversi politiche di accesso e con software eterogeneo– e la problematica che questo comporta.
- **Sicurezza del sistema**: Bisogna definire una politica di accessi alle risorse, la autenticazione e la autorizzazione degli utenti.
- **Uso efficiente delle risorse**: siccome le risorse sono disponibili per vari utenti è necessario definire un meccanismo per ripartire i job in modo automatico ed efficiente. Il software che gestisce questo aspetto è conosciuto come "*middleware*".

[1] Calcolo a griglia.

[2] Ian Foster, Carl Kesselman e Steve Tuecke.

[3] Globus Alliance (`http://www.globus.org/`) è una struttura fatta di organizzazioni ed individui che sviluppano le tecnologie di Grid.

- **Reti di comunicazioni veloci ed affidabili** che rendono possibile la esistenza di Grid.
- **Uso di standard aperti**: Come visto prima, il Grid nasce con la filosofia di condividere risorse. È per questo che l'uso di standard aperti comporta una migliore utilizzazione della struttura creata. Attualmente, gli standard per Grid provengono dal Globus Toolkit[4].

### 1.1.1 Architettura di Grid

La architettura dei sistemi Grid è divisa secondo una struttura a livelli (Figura 1.1 a pagina 4).

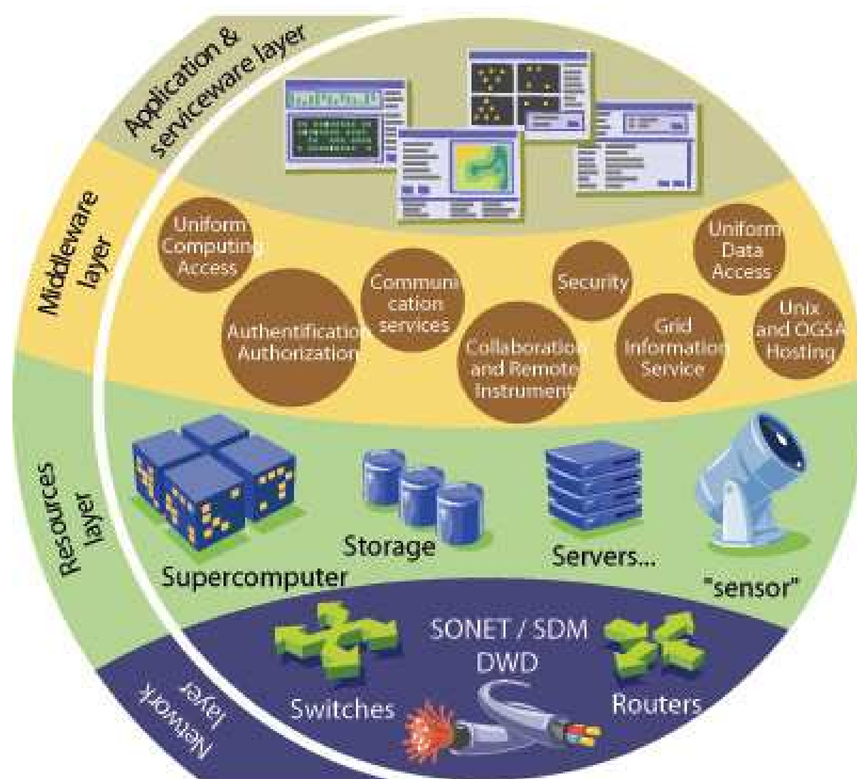



Figura 1.1: Livelli di Grid. (Immagine: CERN)

Nella parte alta della architettura si incontra il **livello applicativo**, il software (che deve essere adattato al Grid) con il quale l'utente si interfaccia. Un livello più in basso,

[4]Struttura di codice aperto sviluppata per la Globus Alliance

il **livello di servizi** (*middleware layer*) fornisce i tools per permettere che i differenti elementi –cioè servers, storage, ecc– vengano usati nell'ambiente Grid, essendo questo livello l'artefice di tutto il funzionamento del Grid.

Sotto il livello di servizi vi è il **livello di risorse**, formato dalle differenti risorse disponibili nella griglia.

Per finire, il **livello di rete** comprende la connettività fra tutte le risorse della Grid.

## 1.2 Worldwide LHC Computing Grid (LCG)

### 1.2.1 Il CERN e il LHC

Il Large Hadron Collider (LHC) è un acceleratore di particele che sta essendo costruito dal CERN colla collaborazione di distinti paesi europei, americani ed asiatici e che diventerà –nella data della sua attivazione, prevista per il 2007– il istrumento scientifico più grande nel mondo.

L'Acceleratore –di 27Km di perimetro e locato vicino a Ginevra, Svizzera– ospiterà quattro sperimenti è si aspetta che produrrà al anno circa di 15 Petabyte di dati. Questi quattro esperimenti vengono conosciuti come ATLAS, ALICE, CMS e LHCb.

Per la naturalezza di questo esperimento bisogna sviluppare un modello computazionale adatto ai requisiti di computo e memorizzazioni di dati. Con questo obbiettivo nasce il World LHC Computing Grid (LCG) sotto la coordinazione del CERN.

### 1.2.2 Il Worldwide LHC Computing Grid

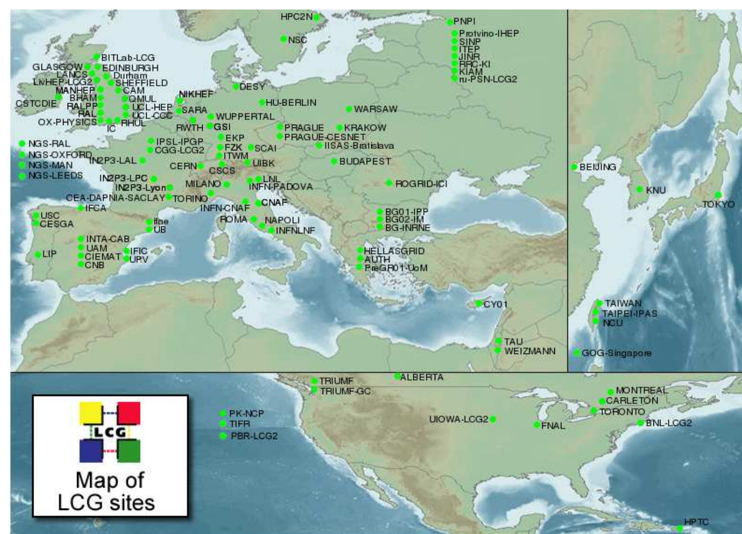



Figura 1.2: Il LCG nel mondo. (Immagine: CERN)

Il LCG ha come scopo costruire e mantenere una infrastruttura di computazione para la memorizzazione e l'analisi dei dati procedenti del esperimento LHC . Questo progetto oggi abbraccia più di XXXXX siti con più di 20000 CPU e YYYYYYYY TB di spazio disco.

C'è bisogna de utilizzare un modello di computazione che garantisca le seguente requisiti:

- Il acceso alle dati a le scientifici interessati in lavorare sul esperimento (circa 15000 persone su tutto il mondo).
- La memorizzazione dei dati durante tutta la vita del esperimento (circa quindici ani).

Per questi due motivi fondamentali, invece di utilizzare una struttura di computazione centralizzata si ha deciso di utilizzare la Grid Computing, con le vantaggi che questo riporta:

- Si garantisce il acceso distribuito alle dati.
- La stessa architettura di Grid permette che la infrastruttura evolva facilmente (replica delle dati e redistribuzione di compiti) nelle situazioni di fallimento di un punto o della costituzione di uno nuovo.
- Il costo de manutenzione di questo sistema rende più semplice, delegando a ogni sito interessato nel esperimento la gestione delle sue risorse.

### 1.2.3 La INFN-Grid

Il progetto INFN-Grid nasce nel seno del Istituto Nazionale di Fisica Nucleare (INFN) nel anno 1999 è diventa il primo progetto di Grid Computing Italiano, costituito da più di venti siti distribuiti nella geografia italiana (Figura 1.3 a pagina 7).

Negli ultimi anni, l'INFN è diventato un importante partner del progetto LCG .

### 1.2.4 Struttura di WLCG

La WLCG è costituita da vari elementi distribuiti geograficamente, e da molti siti (Figura 1.3 a pagina 7) al cui interno si trovano altri elementi (Figura 1.4):

**User Interface (UI)** È la macchina che serve da interfaccia –per sottomettere i job– fra la Grid e l'utente. Può essere un desktop oppure un notebook con un certificato valido installato.

**Resource Broker (RB)** È l'elemento incaricato della sottomissione di job verso un determinato *Computing Element (CE)* , conservando lo stato attuale del job.

Figura 1.3: Struttura geografica INFN-Grid.

**Storage Element (SE)** È il componente di un sito che si occupa della memorizzazione, del acceso e della replica delle informazioni.

**Berkeley Database II (BDII)** Database che memorizza lo stato delle risorse e che viene interrogato nel momento in cui un *Resource Broker (RB)* fa una richiesta.

**Computing Element (CE)** Elemento di controllo e di accesso alle risorse di un sito, capace di sottomettere i job ai *Worker Node (WN)* .

**Worker Node (WN)** Sono le macchine che fanno i calcoli necessari per un determinato job.

Come si può sospettare della enumerazione precedente, gli elementi che devono esistere all'interno di un sito INFI-Grid sono: *Worker Node*, *Computing Element* e gli *Storage Element*, mentre *Resource Broker*, *User Interface* e i *Berkeley Database Information Index* possono essere dislocati nel territorio.

## 1.3 Alta disponibilità

Una serie di servizi nell'ambiente Grid devono essere disponibili almeno al 99%[5]. Inoltre, ci sono servizi che, benché non siano critici, sarebbe desiderabile che abbiano una implementazione ridondante per evitare la perdita dei dati e calcoli fatti fino il punto del fallimento del sistema.

[5]Su base annuale questo significa una indisponibilità minore di 3,6 giorni.

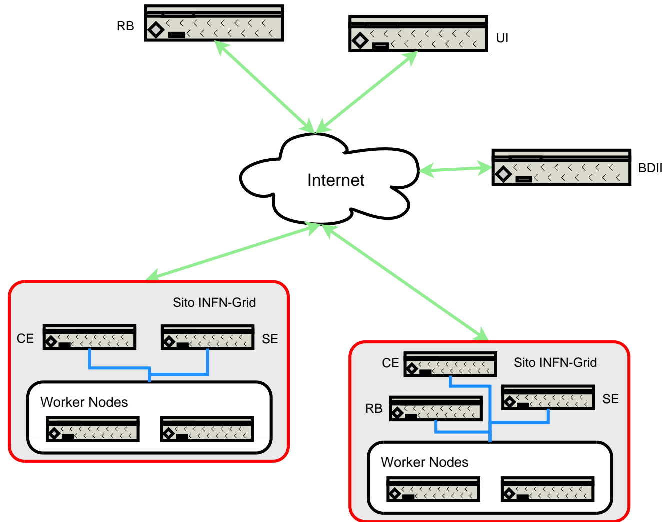



Figura 1.4: Struttura di base di WLCG.

I sistemi informatici, per la sua natura, hanno una grande facilità per fallire, sia per cause hardware –problemi in disco, problemi nella memoria, CPU, ecc– sia per cause software –mala configurazione, driver difettoso, ecc–. Il tempo di risposta davanti di problemi varia secondo la natura del problema, la disponibilità dei tecnici, la possibilità della automatizzazione, ecc; e può andare dei minuti alle settimane.

### 1.3.1 Alta disponibilità tramite backups

La approssimazione più basilare sarebbe quella basata su una soluzione di storage, dove memorizzare i backups delle macchine per essere pronti per riavviare i servizi.

Questa soluzione non è sempre accettabile per parecchi motivi: bisogna avere una persona disponibile 24/7 per ripristinare i servizi; il tempo di ripristino potrebbe essere troppo alto; bisogna avere hardware disponibile per sostituire eventuali pezzi difettosi; i backup possono essere non sufficientemente sincronizzati; ecc.

In questo tipo di soluzione bisogna trovare un compromisi fra la frequenza di backup e la performance dei sistemi che si vogliono affidare: un alto numero di backup ogni poco tempo diminuisce il rendimento; mentre che un numero basso di backup diminuisce la affidabilità.

### 1.3.2 Alta disponibilità tramite ridondanza fisica

Una seconda soluzione è quella della ridondanza delle macchine fisiche, cioè, un mirror –un clone– della macchina che offre il servizio. Questa soluzione, mostrata nella Figura 1.5 a pagina 9, ha come vantaggio che il tempo di ripresa del servizio è minimo –la seconda macchina è un clone funzionante del servizio, dunque bisogna soltanto

cominciare a utilizzare questo clone–, ma offre grandi svantaggi come l'aumento del numero di IP –e host name– nella rete e l'aumento del numero di macchine –con il conseguente costo economico e di manutenzione hardware e software–.

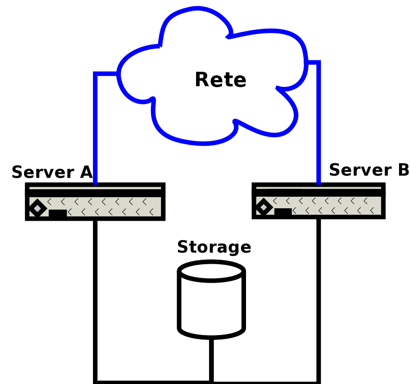



Figura 1.5: Schema della ridondanza fisica.

### 1.3.3 Alta disponibilità tramite virtualizzazione

Una terza soluzione al problema è quella proposta in questa Tesi di Laurea: L'uso di macchine virtuali multiple in esecuzione su alcune macchine fisiche, in modo di poter sostituire quasi immediatamente una macchina virtuale che fallisce; dislocando la esecuzione dei servizi della macchina.

Prima di poter parlare in profondità di questa soluzione, bisogna introdurre cos'è la virtualizzazione (Capitolo 2).

# Capitolo 2

# Virtualizzazione

## 2.1 Cos'è la virtualizzazione

In informatica, la virtualizzazione –cioè l'uso di *Macchine Virtuali (VM)* [1]– è l'uso di un software specifico (chiamato col termino inglese *hypervisor*[2]) per creare distinti ambienti di esecuzione in una macchina fisica (*host*), permettendo di eseguire un sistema operativo (sistema *guest*) diverso in ognuno di questi ambienti.

Ci sono diversi tipi di virtualizzazione, come si vede nella tabella 2.1[3].

| Emulazione | Virtualizzazione | Virtualizzazione a livello di SO | Paravirtualizazzione |
|---|---|---|---|
| Bochs | VMware | OpenVZ | Virtual Iron |
| Qemu | Plex86 | Linux-VServer | User-mode Linux |
| VirtualPC | Microsoft Virtual PC | FreeVPS | L4 |
| DOSEMU | Microsoft Virtual Server | SWsoft Virtuozzo | **XEN** |
| ... | ... | ... | ... |

Tabella 2.1: Tipi di virtualizzazione

## 2.2 Teoria di Popek e Goldberg

Queste teoria stabilisce i requisiti fondamentali che deve riunire un'architettura per essere virtualizzata in forma efficiente. Fu introdotta per Popek e Goldber nel anno 1974 nel suo articolo "*Formal Requirements for Virtualizable Third Generation Architectures*"[4] [5].

[1] A volte si usa la abreviatura del termine inglese *Virtual Machine*, cioè *VM*, invece del termine "macchina virtuale"

[2] Anche conosciuto come *Virtual Machine Monitor*.

[3] È disponibile una tabella più completa nel articolo della Wikipedia: URL: `http://en.wikipedia.org/wiki/Comparison_of_virtual_machines`

[4] Requisiti formali per architetture di terza generazione virtualizzabili.

Un sistema di virtualizzazione – un *hypervisor* oppure *Virtual Machine Monitor (VMM)* – deve:

- Essere equivalente, cioè, un sistema virtualizzato deve avere un comportamento similare al sistema senza virtualizzazione.
- Deve controllare tutte le risorsi del sistema.
- Deve essere efficiente.

Questa teoria stabilisce, per ogni architettura che si vuole virtualizzare, tre insiemi di istruzione e due teoremi che devono soddisfare questi insiemi per adempiere le tre caratteristiche visti previamente.

### 2.2.1 Insiemi di istruzioni

**Privileged** sono quelli che possono interrompere la esecuzione del processore se è in modo di utente e non se è in modo di sistema

**Control sensitive** sono quelli che tentano di cambiare la configurazione delle risorsi nel sistema.

**Behavior sensitive** sono quelli che il suo comportamento oppure il suo risultato dipendono della configurazione delle risorse del sistema.

### 2.2.2 Teoremi

#### 2.2.2.1 Primo teorema

Per qualsiasi calcolatore di terza generazione, un *Virtual Machine Monitor (VMM)* potrà essere costruito sempre che qualsiasi insieme di istruzione *sensitive –control* e *behavior–* è un sub-insieme delle istruzioni privilegiati *–privileged–*.

#### 2.2.2.2 Secondo teorema

Un sistema di terza generazione sarà ricorsivamente virtualizzabile se

1. È virtualizzabile e
2. si può costruire su lui un Virtual Machine Monitor (VMM) senza dipendenze temporali.

### 2.2.3 Effetti di la teoria di Popek e Goldberg

Di conseguenza, per esempio, un'architettura come la System/370 è virtualizzabile perché tutte le sue istruzione sono privilegiati. Una delle architetture più usati –e in cui si basa questa tesi– com'è la x86 non soddisfa i requisiti (ha 17 istruzioni *sensitive*, non privilegiati), quindi non è virtualizzabile.

## 2.3 Tipi di virtualizzazione

### 2.3.1 Emulazione

In questo modello di virtualizzazione, il software –che viene chiamato emulatore– simula per completo l'hardware, permettendo la esecuzione di un software –sia un sistema operativo, sia qualsiasi altro software– senza modifiche. Questo, permette la esecuzione in una architettura –per esempio x86– di un software disegnato per una architettura diversa –per esempio MIPS–.

Un emulatore è tipicamente diviso in diversi moduli:

- Emulatore di CPU.
- Modulo per il sistema di memoria.
- Modulo per i dispositivi di I/O.

### 2.3.2 Virtualizzazione

Bisogna differenziare fra virtualizzazione software e virtualizzazione hardware.

#### 2.3.2.1 Virtualizzazione software (x86)

La virtualizzazione software sarebbe quel modello in cui si simula una parte dell'hardware –soltanto il necessario–, dunque è possibile eseguire un sistema operativo senza modifiche (con la condizione che sia la stessa architettura).

La implementazione della virtualizzazione software per x86 non è una cosa triviale perché questa architettura non soddisfa i requisiti della Teoria di Popek e Goldberg per virtualizzazione (2.2), dunque il rendimento verrà –seriamente– penalizzato.

Esempi di questo modello saranno VMware, Microsoft Virtual PC/Server e la soluzione di software libero Plex86.

#### 2.3.2.2 Virtualizzazione hardware

Ci sono due tecnologie diverse (e incompatibili fra loro) della Intel (chiamata VT) e della AMD (chiamata Pacifica), che includono supporto hardware nei suoi processori per aiutare –diminuendo il costo della emulazione– la virtualizzazione della architettura x86.

Esempi di software che supportano queste tecnologie sono Microsoft Virtual PC/Server, VMWare e Xen (usando la tecnologia VT della Intel per eseguire sistemi operativi senza modifiche).

### 2.3.3 Virtualizzazione a livello di SO

Questa tecnologia consiste nella condivisione di un server fisico in diversi partizioni –*Virtual Enviroments* o *Virtual Private Servers*–, che per gli utenti si comportano come un server reale. In questo modo, la penalizzazione di rendimento, quando c'è presente una grande quantità di partizioni, non è molto elevata.

La implementazione della sicurezza in questo tipo di sistemi diventa un aspetto fondamentale –è il kernel del sistema operativo l'incaricato della gestione delle partizioni, dunque senza una buona gestione delle risorse, è facile causare un *Denial of Service*–.

Nella Tabella 2.1 ci sono alcuni esempi di questa tecnologia.

### 2.3.4 Paravirtualizzazione

In questo modello non si simula l'hardware, ma invece si offre una API speciale –simile ma non identica al hardware esistente– al SO *guest*, che deve essere modificato per utilizzare questa API.

Questa implementazione comporta due caratteristiche fondamentali ed interessanti rispetto alle altre soluzioni:

- L'*hypervisor* diventa molto più semplice.
- Le macchine virtuali che girano su questo sistema hanno un rendimento maggiore.

In questo gruppo ci sono varie soluzioni, come User-mode Linux (UML), Denali, L4, Virtual Iron e quella oggetto di questa tesi: **Xen** (Vedere Sezione 2.4).

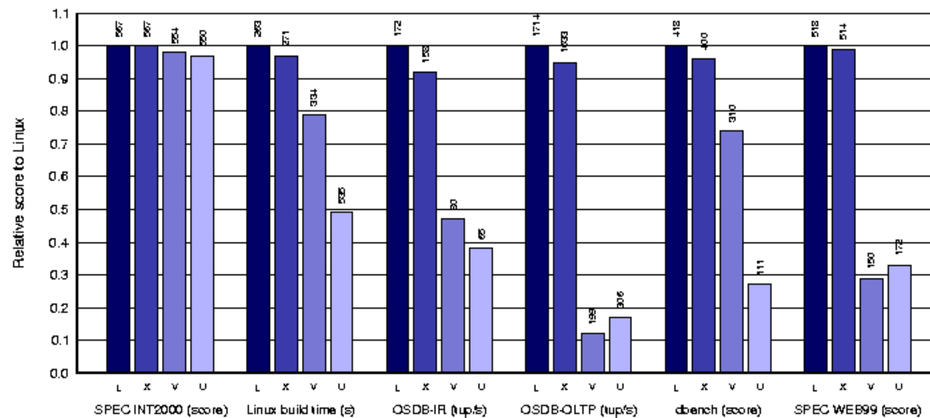
## 2.4 Xen

In questa sezione si parlerà soltanto delle caratteristiche di Xen, la sua architettura, ecc. Per una spiegazione di come utilizzarlo, guardare l'Appendice A.

### 2.4.1 Cos'è Xen

Secondo il sito web del progetto Xen: "*Xen è un* virtual machine monitor *–cioè, un* hypervisor*– per x86 che supporta la esecuzione di distinti sistemi operativi ospite con un rendimento senza precedenti e isolamento di risorsi. Xen è Software Libero, sotto i termini della GNU General Public License"* [7].

Come viene detto, Xen è un *hypervisor,* sviluppato come Software Libero per la Università di Cambrigde, che utilizza la tecnica chiamata paravirtualizzazione (2.3.4), ottenendo un alto rendimento (circa un 8% meno che il sistema operativo nativo) in una architettura –x86– sempre penalizzata nella virtualizzazione. Nella Figura 2.1 a pagina 15 è possibile osservare che la diminuzione del rendimento contro altri sistemi di virtualizzazione è minima.




Native Linux (L), Xen/Linux (X), VMware Workstation 3.2 (V), User Mode Linux (U).

Figura 2.1: Confronto del rendimento (Immagine: Xen)

Ci sono due versioni di Xen con parecchie differenze e incompatibilità tra loro: Xen 2.0 e Xen 3.0. Lo sviluppo della versione 2.0 non quasi continuato, quindi attualmente la versione in cui si sta lavorando è **Xen 3.0**. Da qui in avanti, sempre se si parla di Xen (senza aggiungere il numero di versione), sarà in referenza a Xen 3.0.

### 2.4.2 Paravirtualizzazione in Xen

Xen introduce modifiche sia nelle macchine *host* che nelle macchine *guest*.

La macchina in cui verrà fatta la virtualizzazione (cioè, la macchina *host*) non sarà più una macchina x86: diventa una macchina con architettura Xen-x86 e i sistemi operativi che si vogliono virtualizzare devono essere adattati per questa architettura.

È per questo che non tutti i sistemi operativi hanno supporto per essere virtualizzati tramite Xen, e che non tutti possono eseguire Xen come *hypervisor*. La Tabella 2.2 a pagina 15 mostra gli sistemi operativi compatibili con Xen 3.0 al momento di scrittura della tesi.

| Sistema operativo | Host | Guest |
|---|---|---|
| Linux 2.6 | Si | Si |
| NetBSD 3.0 | No | In alto grado di sviluppo |
| FreeBSD 5.3 | No | In alto grado di sviluppo |
| Plan9 | No | In sviluppo |
| ReactOS | No | Proggettato. |
| S.O. senza modifiche | No | Supporto iniziale tramite Intel VT |

Tabella 2.2: Compatibilità tra Xen 3.0 e distinti sistemi operativi

In questa tesi, si parlerà di Xen utilizzando sempre come *host* un kernel Linux della serie 2.6.

### 2.4.3 Architettura di Xen

La architettura x86 ha un modello di protezione basato in quattro livelli[5] (numerati da zero a tre) di privilegi. Nella Figura 2.2 a pagina 16 si può osservare un confronto fra un sistema operativo normale e un Xen.

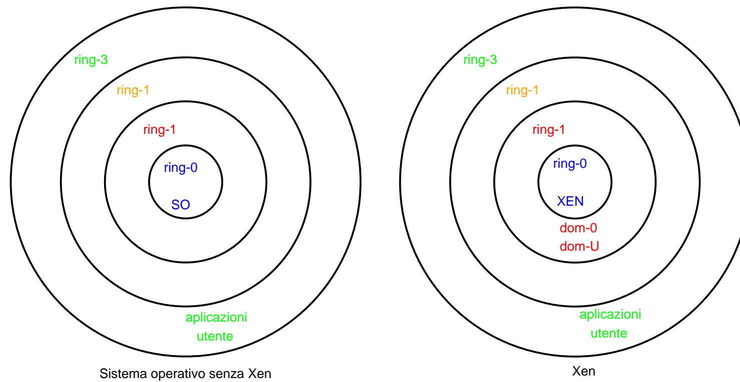



Figura 2.2: Livelli di privilegi

Un sistema operativo qualsiasi avrà la seguente struttura:

**ring-0** Livello dove il kernel del sistema operativo si esegue. Questo livello è l'unico dove si possono invocare certo tipo di istruzioni (privilegiati).

**ring-1, ring-2** Non usati eccetto per OS/2.

**ring-3** Livello dove si eseguono l'applicazioni utente.

Invece, un sistema operativo modificato per eseguire Xen come *hypervisor* avrà la seguente:

**ring-0** Livello dove si carica Xen.

**ring-1** Livello dove si caricano le *Macchine Virtuali (VM)* .

**ring-2** Non usato.

**ring-3** Livello dove si eseguono l'applicazioni utente.

Una volta installato Xen in una macchina fisica, è caricato nel ring-0 e poi avvia automaticamente una prima macchina virtuale –chiamata *VM0* oppure dom-0– in forma trasparente per l'utente (infatti, questa macchina virtuale ha come sistema operativo il

[5] chiamati col termine inglese: ring

sistema sul cui si ha installato Xen). Il *dom-0* ha privilegi speciali: può accedere all'hardware direttamente e può creare, distruggere, mettere in pausa, migrare, ecc. altri macchine virtuali (chiamati dom-U).

In Figura 2.3 a pagina 17 è rappresentato lo schema a blocchi dell'architettura Xen. Il *dom-0* è l'unico che può accedere all'hardware direttamente –tramite i driver nativi–, esportando i "*backend drivers*" ai "*frontend drivers*" presenti nei distinti *dom-U*. Perciò, tutte le macchine virtuali create a posteriori comunicano con l'hardware tramite il *dom-0* .

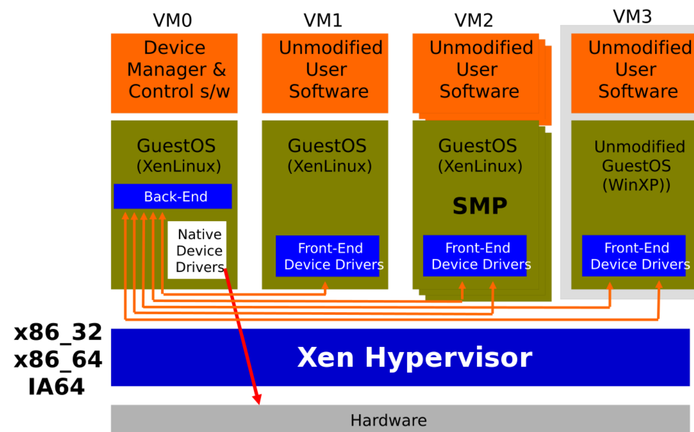



Figura 2.3: Architettura di Xen

### 2.4.4 I daemon di Xen

#### 2.4.4.1 Xen daemon: xend

Per controllare e creare i distinti *dom-U* e poter fare differenti azioni su loro, nel *dom-0* deve girare un processo speciale in secondo piano (*daemon*), chiamato xend.

Questo processo è l'incaricato di ricevere diversi ordini[6] (tramite il comando `xm`, tramite una interfaccia web, ecc).

#### 2.4.4.2 Xen Store Daemon: xenstored

Questo daemon è un server –che si esegue nel *dom-0*– che memorizza le informazioni delle differenti macchine virtuali, in un database –condiviso fra i differenti *dom-U*– in forma di albero. Ha come oggetto memorizzare informazione di e servire come controllo delle differenti macchine virtuali che sono in esecuzione.

### 2.4.5 Caratteristiche di Xen

- Virtualizzazione con una penalizzazione nel rendimento molto bassa.

[6] Si parlerà di questi ordini e del funzionamento di Xen nel Appendice A.

- Possibilità di mettere in pausa la esecuzione delle Macchine Virtuali (VM) .
- Migrazzione delle VM senza interrupzioni –*live migration*–.
- Realocazzione di memmoria in esecuzione.
- Controllo delle VM via interfaccia web.
- È software libero (licenza GNU General Public License (GPL) ).

## 2.5 Vantaggi della virtualizzazione

La virtualizzazione –e più concretamente la virtualizzazione tramite Xen– permette di:

- Eseguire distinti sistemi operativi contemporaneamente in una singola macchina.
- Separare i servizi dall'hardware (per quelli che non dipendono di un HW determinato) e dal sistema operativo installato sull'hardware.
  - Una macchina virtuale potrà girare in qualsiasi macchina fisica.
  - Il software utilizato nella struttura LCG non è sempre aggiornato all'ultima versione e quindi potrà non avere supporto per harware recente. Le macchine virtuali fano sparire questo problema.
- Isolare le macchine su cui sono in esecuzione i servizi, in modo che, per esempio, ogni utente disponga di una macchina completa per il suo uso (più sicurezza davanti a intrusioni).
- Possibilità di clonare le macchine per fare test e aggiornamenti, senza compromettere la integrità della macchina originale, con la possibilità di tornare indietro in maniera controllata.
- Possibilità di scalabilità, entro certi limiti.
- Mettere in pausa le macchine con la possibilità di migrare una macchina virtuale a un'altra macchina fisica e riprendere l'esecuzione nel punto di arresto. Questo permette:
  - Load-balancing: si possono migrare macchine virtuali di un *host* con un alto carico a un *host* senza carico senza fermare le macchine (*live-migration*).
  - Alta disponibilità: se una macchina fallisce si possono migrare le macchine virtuali (prima che fallisca) o recuperare un snapshot e farlo eseguire su un'altra macchina *host*.

## 2.6 Alta disponibilità tramite virtualizzazione

Una terza approssimazione al problema sarebbe quella proposta in questa Tesi di Laurea: Il uso di multiple macchine virtuali girando su ogni macchina fisica, dislocando la esecuzione dei servizi della macchina.

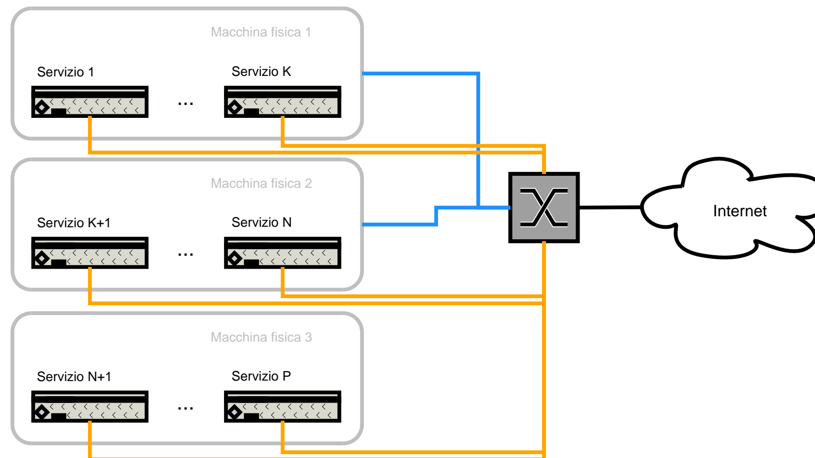



Figura 2.4: Struttura di rete con macchine virtuali.

In questo modo, utilizzando la struttura di macchine virtuali e di un sistema automatico di monitoraggio e controllo, si potrebbe ottenere parecchie vantaggi rispetto alle altre soluzioni:

1. Riduce il downtime quasi sempre a pochi secondi.
   (a) Avviare una nuova macchina virtuale in un'altra macchina fisica e –quasi– istantaneo.
   (b) Se ci sono problemi in una macchina si può migrare a un'altra prima di che la macchina fallisca, senza fermarla. Downtime=0.
2. Permette facilmente lo sviluppo ed il test di versioni diverse, isolando la esecuzione delle nuova versione.
3. In linea di principio rende indipendenti dall'hardware sottostante i servizi e la installazione; dunque,
4. si potrebbe definire una *Macchine Virtuali (VM)* tipizzata per servizi generici e distribuirla su tutte le macchine.

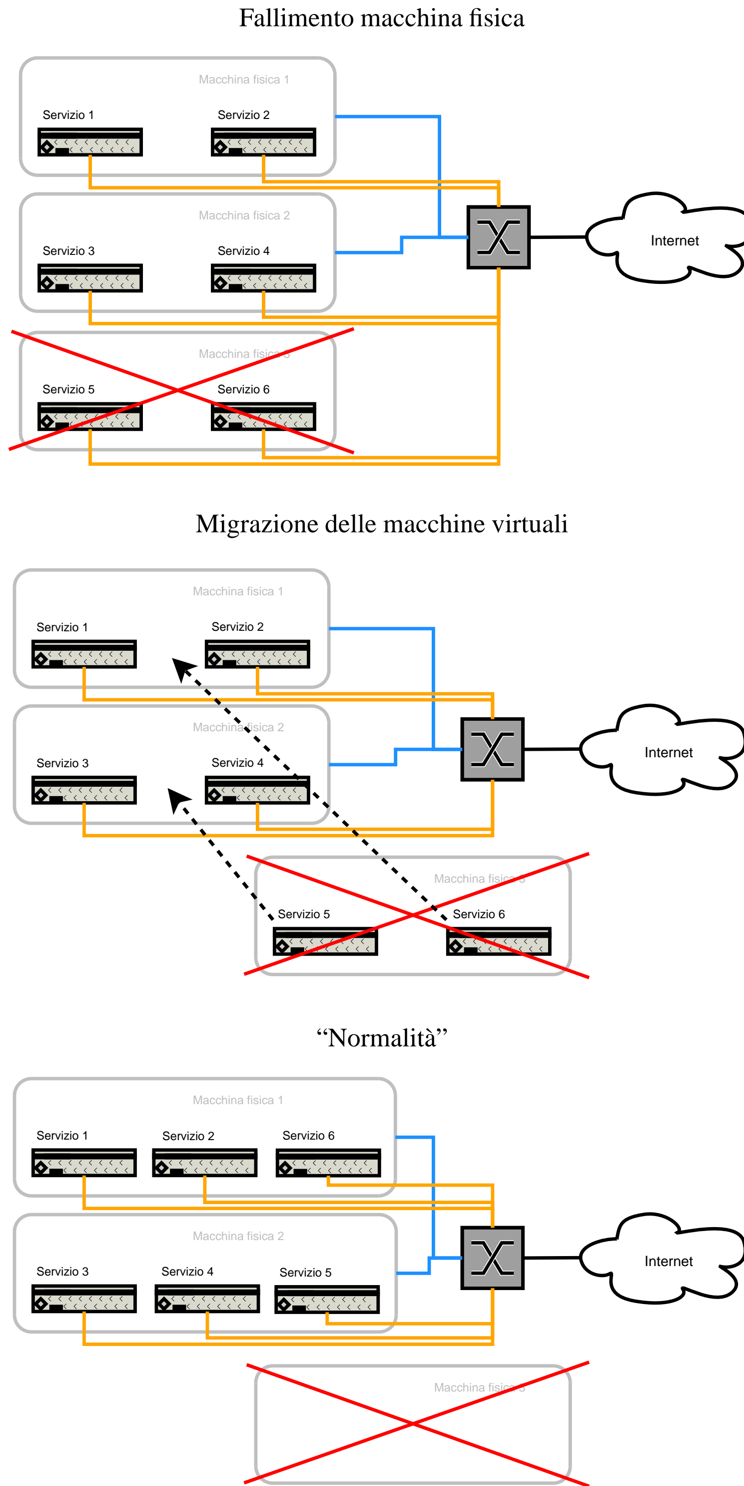



Figura 2.5: Ripristino di due servizi virtualizzati

# Capitolo 3

# Il prototipo: analisi delle possibilità e test delle componenti

In questo capitolo, si spiegherà la teoria su cui si baserà il prototipo per fare funzionare i servizi di Grid, di forma altamente affidabile, tramite l'uso di macchine virtuali e inoltre si parlerà dei test che sono stati fatti per scegliere la migliore soluzione.

## 3.1 Struttura

La struttura proposta è quella mostrata nella Figura 3.1 a pagina 22.

Gli elementi presenti –e le loro funzioni– in questa soluzione sono:

**Macchine fisiche** In questo modello le macchine fisiche presenti hanno soltanto la funzione di ospitare una o più macchine virtuali, e quindi devono eseguire Xen. Per questo, in queste macchine si può utilizzare qualsiasi sistema GNU/Linux compatibile con i requisiti di Xen.

**Macchine virtuali** Tutti si servizi che vogliono approfittare la alta disponibilità devono essere virtualizzati. Nel modello finale, sono queste macchine –le virtuali– quelle incaricate di offrire i servizi alla Grid, quindi c'è bisogno di utilizzare la distribuzione di GNU/Linux utilizzata originalmente, cioè, Scientific Linux.

**Storage** I filesystem su cui si eseguono le macchine virtuali. Le immagine delle macchine vengono scaricate da un server, tramite rete.

**Servizio di healtcheck** Questo servizio deve essere eseguito sia in una macchina fisica, sia in una macchina virtuale, siccome è incaricato di gestire e verificare il

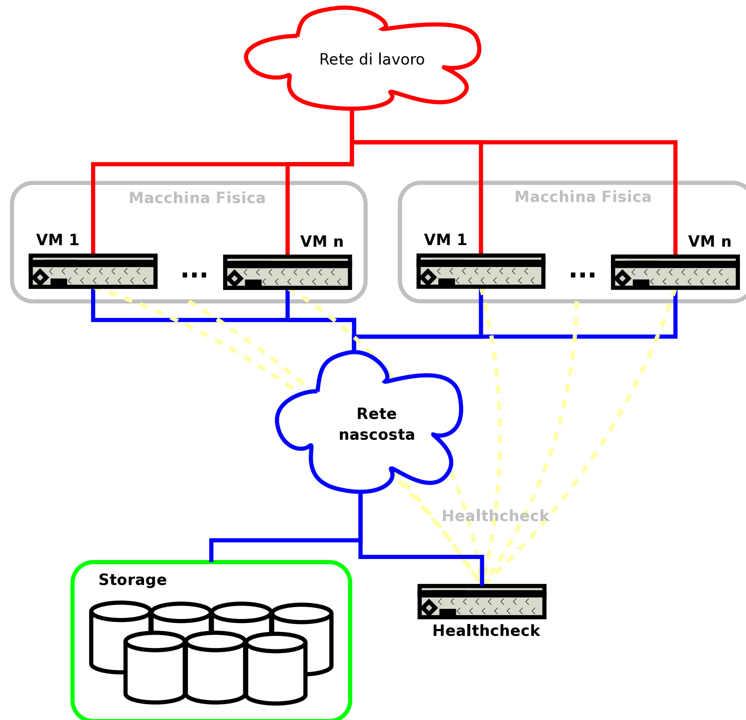



Figura 3.1: Prototipo di alta disponibilità tramite Xen.

funzionamento del sistema. Nella Sezione 3.5 si parlerà più in detaglio di questo sistema.

In questo schema, le macchine virtuali –cioè, le macchine dove si eseguono realmente i servizi– diventano indipendenti dall'hardware sottostante, con i vantaggi che questo comporta: ogni macchina virtuale potrà essere eseguita in ogni nodo fisico, bisogna soltanto portare la sua immagine –il suo filesystem– da una macchina all'altra.

Bisogna quindi utilizzare un server di storage per caricare le immagini tramite la rete e così potere spostare la macchina di un *host* automaticamente (tutte le macchine vedono tutte le immagini di tutte le Macchine Virtuali (VM) ).

## 3.2 Soluzioni distinte per lo storage

Uno degli elementi fondamentali di questo prototipo è il sistema utilizzato per memorizzare e distribuire le immagini delle macchine virtuali. Per questo, bisogna fare uno studio delle distinte tecnologie e soluzioni esistenti sul mercato per rendere possibile la distribuzione di un filesystem via rete.

Queste si dividono in diversi gruppi: Block device remoti via hardware, block device remoti via software[1] e filesystem distribuiti.

### 3.2.1 Block device remoti via hardware

Queste soluzioni si basano sulla esportazione di diversi block device da varie dispositivi collegati in una rete. In particolare, sono quelli basati su un hardware dedicato, di solito costoso, e specificamente disegnato per fare questa funzione.

La tecnologia più conosciuta è *Fibre Channel*.

#### 3.2.1.1 Fibre Channel

Fibre Channel (FC) è una tecnologia di rete di alta velocità, utilizzata principalmente per lo storage in rete. FC diventa importante e fondamentale nell'ambito dei grandi supercomputer e in ambito aziendale. Ci sono diverse topologie (Figura 3.2 a pagina 23):

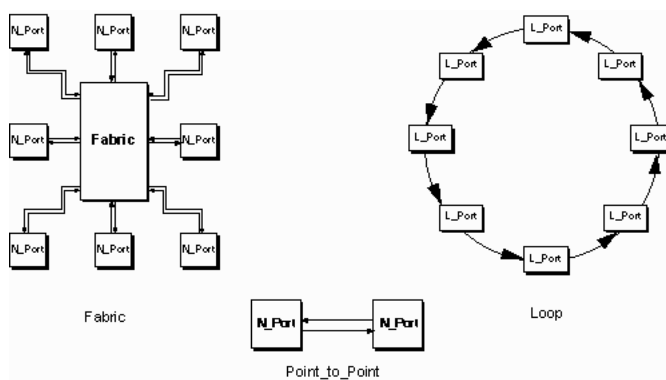



Figura 3.2: Le topologie FC.

**Switched Fabric** Tutti i device sono collegati ai FC switches, di forma similare a quelli Ethernet.

**Point-to-point** Due dispositivi sono collegati direttamente. Diventa la soluzione più semplice e limitata.

**Arbitrated Loop** Tutti i device sono collegati in forma di anello, di forma similare alla tecnologia Token Ring; è la più sensibile ai problemi.

A dispetto del suo nome, FC può essere implementato sia con cavi di fibra ottica, sia con cavi twisted-pair[2]. Il protocollo utilizzato per la gestione dei block device è il protocollo SCSI[3].

[1] Di solito questi due sistemi si chiamano Storage Area Network (SAN) .

[2] I cavi twisted-pair sono dei cavi dove 2 conduttori sono incrociati per diminuire le interferenze elettromagnetiche.

[3] Small Systems Computer Interface (SCSI) .

### 3.2.2 Block device remoti via software

Queste soluzioni sono similari ai block device remoti via hardware, non si utilizza un hardware specifico ma un software speciale. Queste soluzioni sono una buona alternativa all'uso di una soluzione hardware, in quanto il prezzo della implementazione non è cosi elevato e le prestazioni sono abbastanza buone.

Le due soluzioni principali sono GNBD e iSCSI.

#### 3.2.2.1 iSCSI

L'Internet SCSI (iSCSI) [4] è un protocollo definito nel Request For Comments (RFC) 3720[14] dall'Internet Engineering Task Force (IETF) [5] per utilizzare il protocollo Small Systems Computer Interface (SCSI) , usando come mezzo di trasporto una rete TCP/IP (altri mezzi di trasporto potrebbero essere Fibre Channel, InfiniBand, USB (Universal Serial Bus), IEEE 1394 (FireWire), ecc.). La vasta diffusione delle reti TCP/IP e lo sviluppo della tecnologia Gigabit Ethernet[6] fa si che le Storage Area Network (SAN) [7] basate su iSCSI siano una soluzione economica, ma non per questo poco efficiente.

iSCSI è basata su un'architettura (Figura 3.3 a pagina 25) client-server (quella di SCSI ), dove il client viene chiamato "*initiator*" mentre il server viene chiamato "*target*". In un sistema iSCSI , un *initiator* fa una richiesta di un servizio a un *target* utilizzando uno dei protocolli di trasporto definiti per lo standard SCSI .

Attualmente ci sono varie implementazione di iSCSI per i sistemi GNU/Linux:

*Initiator* La implementazione più stabile e la più matura è quella chiamata "*core-ISCSI*" `http://www.kernel.org/pub/linux/utils/storage/iscsi/`.

*Target* La implementazione su cui si sta lavorando di più è "*The iSCSI Enterprise Target*" `http://iscsitarget.sourceforge.net/`.

L'installazione e configurazione di iSCSI viene descritta nell'Appendice B.

#### 3.2.2.2 GNBD

Global Network Block Device (GNBD) consiente l'accesso a distinti block device – sia block device locali oppure block device in un SAN – attraverso una rete che di solito è Ethernet. È una tecnologia sviluppata per la Red Hat per essere utilizzata –originalmente– con Red Hat Global File System (GFS) .

L'unica distribuzione esistente è quella della Red Hat, che è parte di una suite conoscita come Cluster `ftp://sources.redhat.com/pub/cluster/releases`.

[4] *Internet Small Computer Systems Interface* oppure *Internet SCSI*.
[5] `http://www.ietf.org/`.
[6] Tecnologia per implementare reti Ethernet a una velocità nominale di 1 Gigabit per secondo.
[7] Storage Area Network, soluzione di storage in reti usando protocolli di basso livello (SCSI, ATA, ecc).

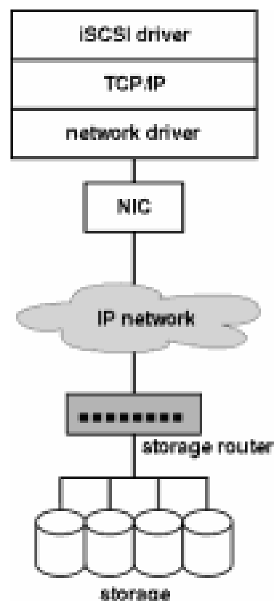



Figura 3.3: Architettura di iSCSI.

### 3.2.3 Filesystem distribuiti

I filesystem distribuiti più utilizzati e conosciuti sono due: **GFS** , sviluppato dalla Red Hat e **General Parallel File System (GPFS)** , sviluppato dalla IBM. Tutti e due permettono accedere da un numero indeterminato di calcolatori –originalmente quelli che formano un cluster– allo stesso filesystem.

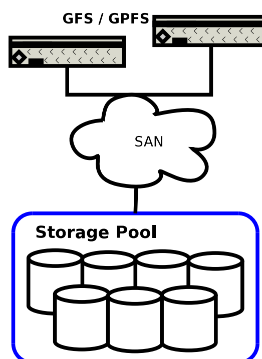



Figura 3.4: Schema di filesystem distribuiti.

Questi sistemi distribuiti permettono:

- Una alta scalabilità e flessibilità.
  - Un aggiornamento unico del software per tutte le macchine.

- Una ottima gestione di grandi volumi di dati (vengono usati come un'unica partizione).

- Riducono l'uso di copie ridondanti di dati.

## 3.3 Test di compatibilità

### 3.3.1 Test di sistemi *host*

I test fatti si sono basati sulla installazione e prova di Xen 3.0.2 (in una prima installazione è stato usato Xen 3.0.1) su varie distribuzioni GNU/Linux come sistema *host*, sempre utilizzando il kernel Linux 2.6.16-xen (nella installazione di Xen 3.0.1 si è stato usato il kernel Linux 2.6.12.6-xen):

- Slackware Linux 10.2.

- Gentoo Linux 2006.0.

- Scientific Linux.

- Ubuntu Drapper.

- Fedora 5.

Tutte quante le installazioni sono state pulite e senza errori speciali (bisogna soltanto guardare con attenzione i requisiti della installazione di Xen).

### 3.3.2 Test di sistemi *guest*

Si sono fatti i test di compatibilità per l'uso come sistema *guest* di Scientific Linux (nelle versione 3 e 4), sempre utilizzando il kernel Linux modificato per Xen (come viene detto, Linux 2.6.16-xen per la versione di Xen 3.0.2 e Linux 2.6.12.6-Xen per la versione 3.0.1).

Si è verificata l'indipendenza delle macchine virtuali dall'hardware sottostante, eseguendole (e anche facendo migrazione) su parecchie macchine con hardware diverso. L'unico aspetto da ricordare è se l'architettura sia a 32 bit oppure a 64 bit.

### 3.3.3 Test di dispositivi storage

Si è fatta anche la verifica della compatibilità tra i driver delle distinte soluzioni di storage disponibili e il kernel Linux modificato da Xen; funzionano tutti senza problemi.

## 3.4 Test di I/O

Si sosno rializzato diversi test per alcune delle soluzioni mostrate nella Sezione 3.2, sia in macchine a 32 bit, sia in macchine a 64 bit per scoprire la soluzione migliore per essere usata in questo prototipo.

### 3.4.1 IOzone

IOzone[8] è uno strumento ampiamente utilizzato per fare prove e comparative di rendimento *–benchmark–* nel accesso al'disco (sia tra differenti filesystem, sia tra diffreteni dispositivi).

Si sono fatti test tramite l'uso di questo tool, sottomettendo le macchine a un alto carico nei giorni che sono durate le prove. Si è verificata anche la stabilità delle soluzione di storage e di Xen (migrazione delle macchine, live-migration, pausa delle macchine) sempre sotto un alto carico di lavoro e un alto tasso di I/O.

### 3.4.2 Caratteristiche delle macchine utilizzate nei test

Le macchine fisiche utilizzate (in totale sono due) per fare dei test sono sempre le stesse, con le caratteristiche mostrate nella Tabella 3.1 a pagina 27. Le macchine virtuali caricate su queste macchine (sempre due per macchina) hanno una memoria RAM di 512MB e eseguono una Scientific Linux 4.2 con un Kernel 2.6.16-xen.

| Processore | Dual AMD Opteron 2GHz |
|---|---|
| Memmoria | 1GB |
| Disco | 40GB |
| Conessione Rete | Gigabit Ethernet / Fibra ottica |
| Sistema Operativo | Gentoo Linux - Kernel 2.6.16-xen |

Tabella 3.1: Caratteristiche macchine fisiche.

Per gli altri test, si è utilizzato il fileserver descritto nella Tabella 3.2 a pagina 27.

| Processore | Dual Pentium III 1GHz |
|---|---|
| Memmoria | 256MB |
| Disco | 210GB - RAID 5 |
| Conessione Rete | Gigabit Ethernet |
| Sistema Operativo | Slackware Linux 10.2 - Kernel 2.6.15.6 |

Tabella 3.2: Caratteristiche fileserver.

In alcuni test si è usato anche un disco FC con le caratteristiche della Tabella 3.3 a pagina 28.

| Spazio disco | 4797776,28MB |
|---|---|
| Conessione Rete | Fibra Ottica |

Tabella 3.3: Caratteristiche disco FC.

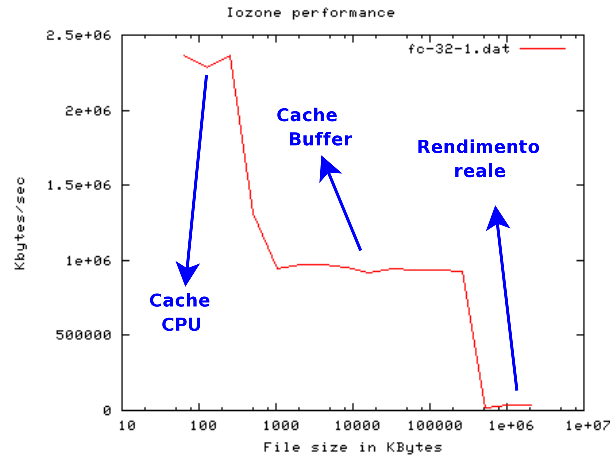



Figura 3.5: Esempio di risultato di un test con IOzone.

### 3.4.3 Interpretazione dei risultati

Prima di introdurre i test di I/O fatti sulle soluzioni di storage, bisogna spiegare come verrano presentati. Nella Figura 3.5 a pagina 28 c'è un esempio di un test fatto con IOzone. Nell'asse $X$ c'è la dimensione del file a cui si accede, mentre nel asse $Y$ c'è la velocità di accesso.

Si può vedere che ci sono tre zone differenziate:

- Zona alta: Queste misure sono il risultato dell'accesso al'disco, quando il file è molto piccolo ed esiste la influenza della cache della CPU.
- Zona media: Queste misure vengono influenzate dalla cache del buffer.
- Zona bassa: In questa zona è dove si trovano le misure della velocità di accesso al'disco quando non ci sono effetti di nessuna cache, cioè, questa è la zona dove stanno i risultati reali del disco.

### 3.4.4 Risultati

#### 3.4.4.1 Fibre Channel

Si è utilizzato un disco FC con le caratteristiche della Tabella 3.3 a pagina 28, collegato alle due macchine fisiche descritte previamente (tramite una connessione in fibra ottica a 2Gb/s).

[8]URL: http://www.iozone.org/

I risultati ottenuti si possono vedere nelle Figure 3.6-3.9.

Si può osservare un piccolo migliore rendimento –più visibile nella Figura 3.6 a pagina 29– per file di dimensioni sotto 1KB nell uso di FC con un sistema operativo a 64 bit, mentre che per il resto dei test il rendimento è sempre circa lo stesso per tutte e due le architetture.

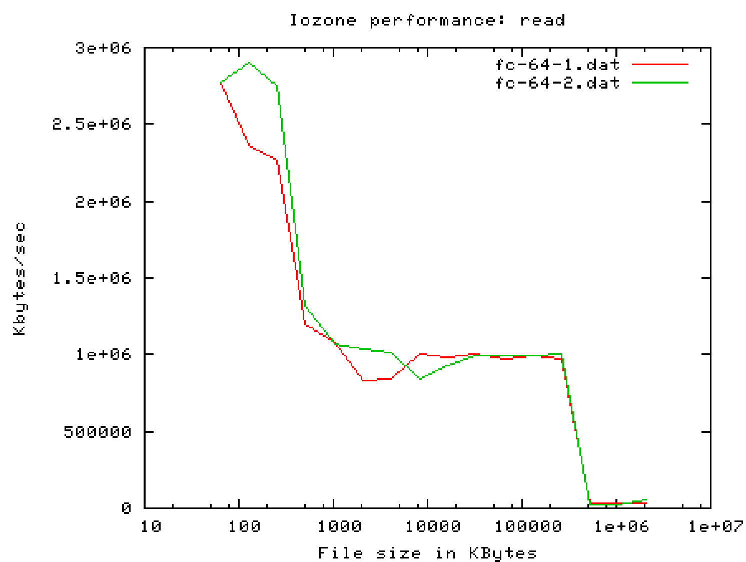



Figura 3.6: Test di lettura a 64 bit.

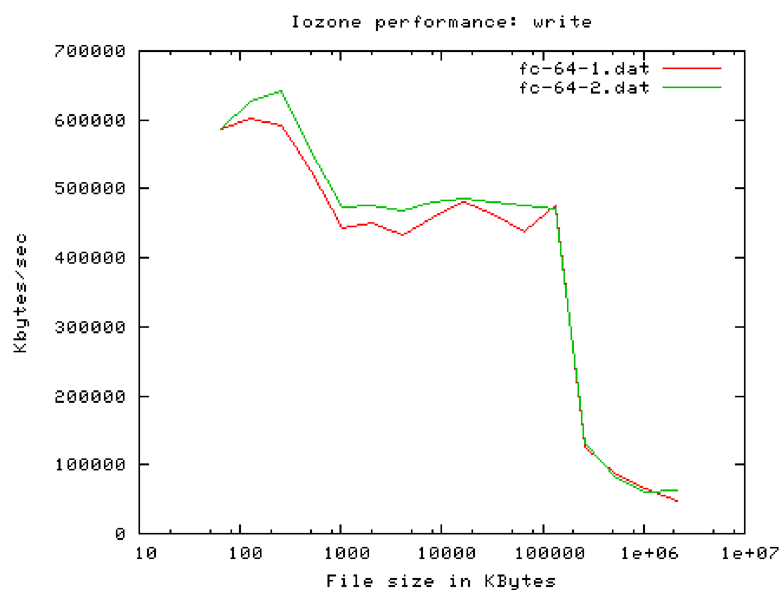



Figura 3.7: Test di scrittura a 64 bit.

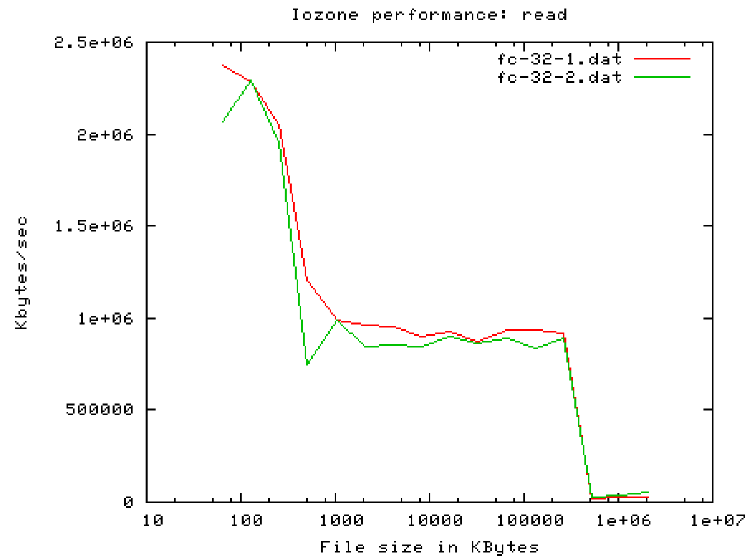



Figura 3.8: Test di lettura a 32 bit.

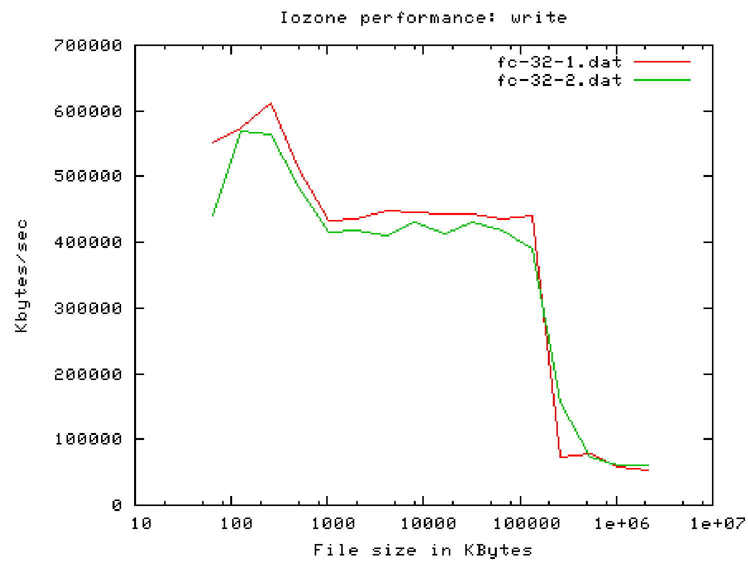



Figura 3.9: Test di scrittura a 32 bit.

#### 3.4.4.2 GNBD

In questi test si è utilizzato il fileserver descrito nella 3.2 e le due macchine nella 3.1. Si sono utilizzati due macchine virtuali contemporaneamente sullo stesso fileserver.

I risultati si possono vedere nelle Figure 3.10-3.13.

In queste figure si può osservare un comportamento che non è normale, proabil-

mente a causa di la esistenza di carico nella rete. Ad esempio, nella Figura 3.12 a pagina 32, nelle zone alta e bassa si vede una differenza di velocità molto elevata che non dovrebbe esistere.

Inoltre, come nella soluzione FC , si può osservare un incremento del rendimento in file di piccole dimensioni per la architettura a 64 bit.

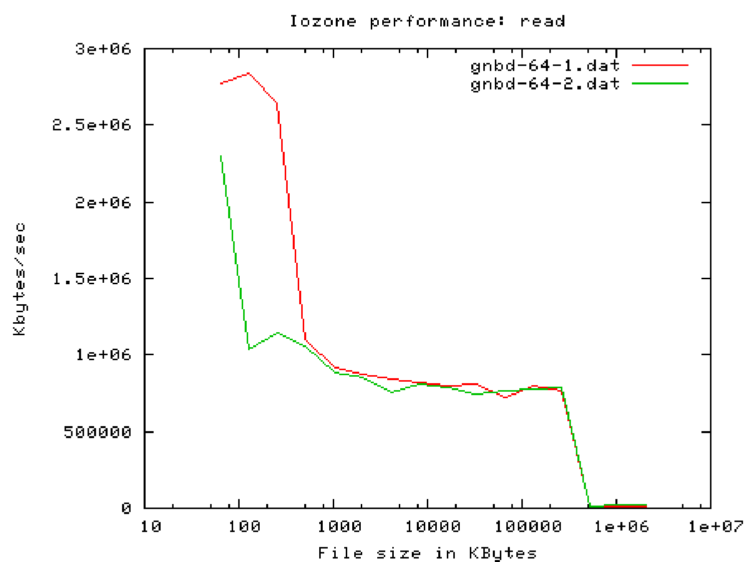



Figura 3.10: Test di lettura a 64 bit.

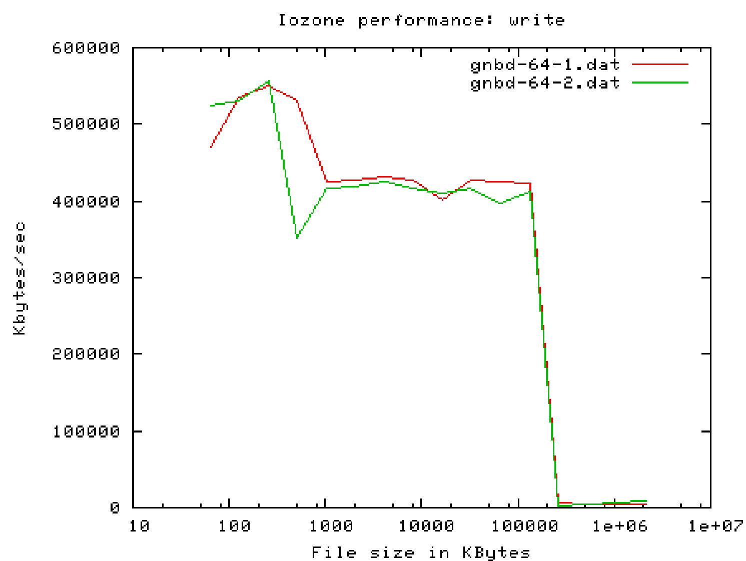



Figura 3.11: Test di scrittura a 64 bit.

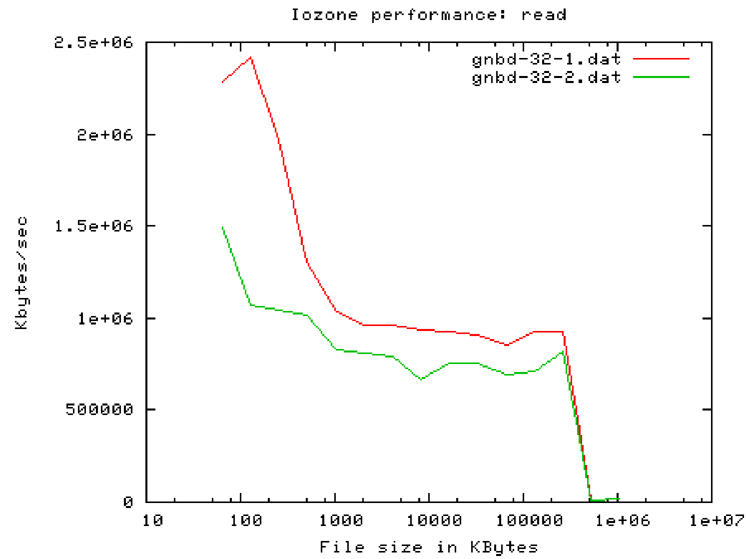



Figura 3.12: Test di lettura a 32 bit.

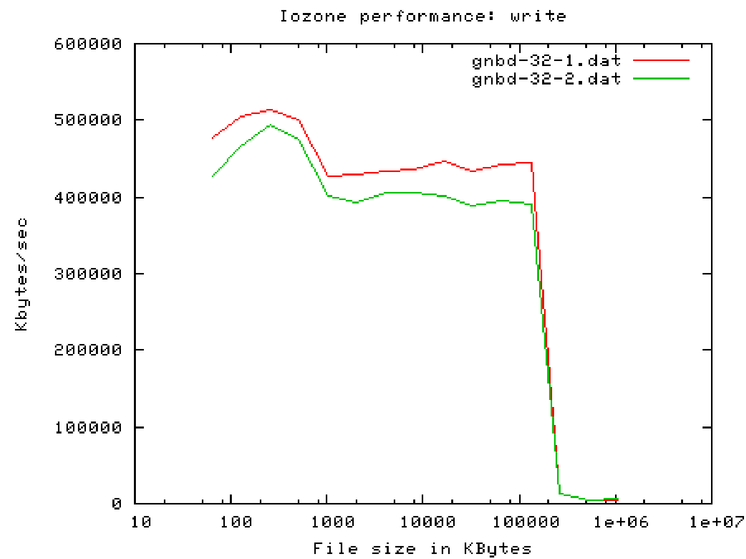



Figura 3.13: Test di scrittura a 32 bit.

#### 3.4.4.3 iSCSI

Per questi test si è utilizzato sempre lo stesso fileserver utilizzato per i test con GNBD, già mostrato nella Tabella 3.2 a pagina 27 utilizzando come *initiator* la implementazione chiamata ”*core-ISCSI*” [16]. Nelle macchine fisiche (Tabella 3.1 a pagina 27) si è utilizzato come *target “The iSCSI Enterprise Target”* [18]. Si sono realizzati i test con

due macchine virtuali contemporaneamente sullo stesso fileserver. I risultati si possono vedere nelle Figure 3.14-3.17.

C'è un comportamento analogo a GNBD –cioè, una diferenza nei risultati fra i due test–, probabilmente per causa della conesione di rete (il disco FC ha una conessione dedicata, mentre il fileserver ha una conessione condivisa). Anche si può osservare il aumento del rendimento nella architettura a 64 bit per file piccoli.

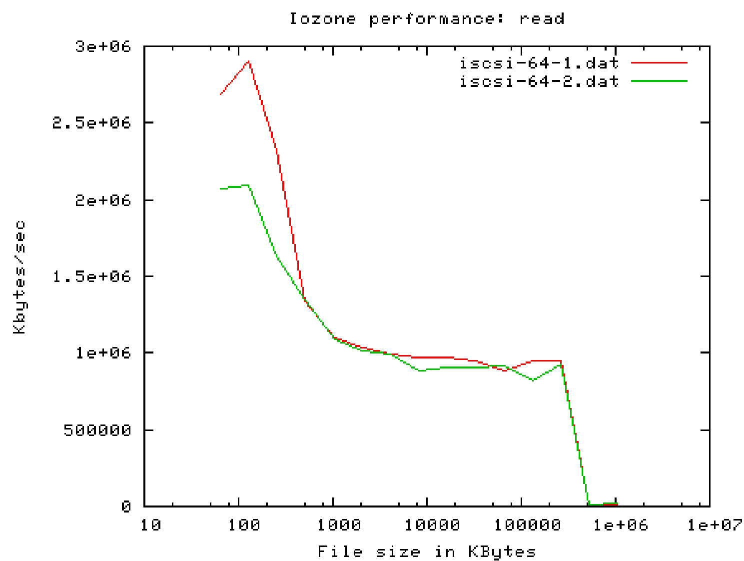



Figura 3.14: Test di lettura a 64 bit.

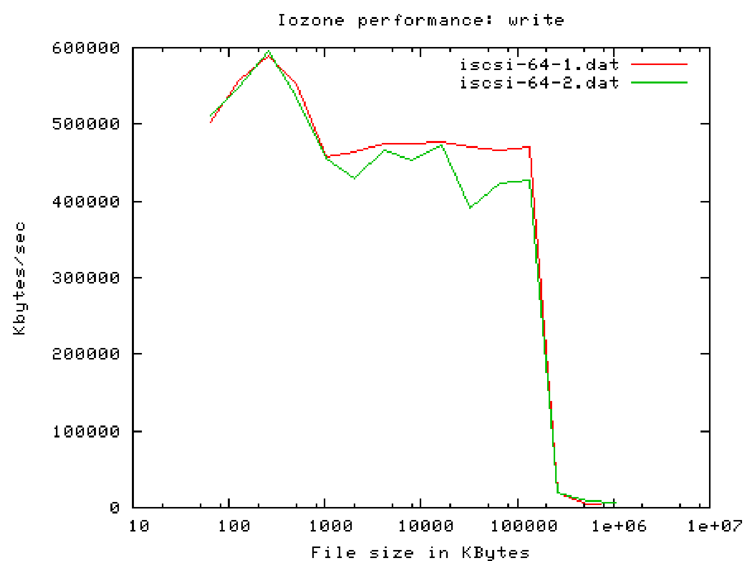



Figura 3.15: Test di scrittura a 64 bit.

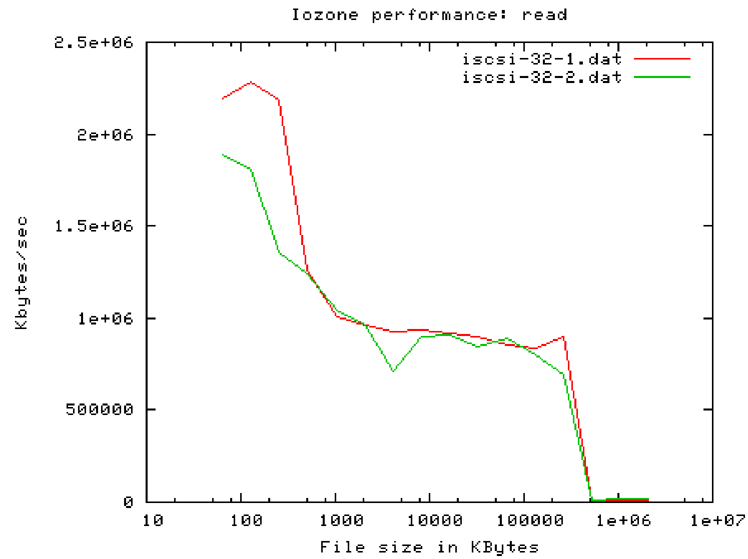



Figura 3.16: Test di lettura a 32 bit.

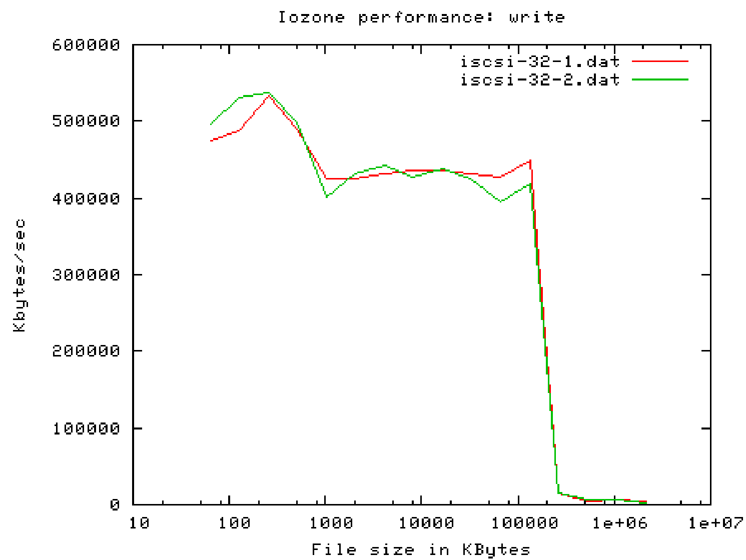



Figura 3.17: Test di scrittura a 32 bit.

### 3.4.5 Confronto risultati

In questi confronti soltanto si ha utilizzato una porzione dei dati ottenuti significativa per il rendimento reale dil sistema di storage, cioè, si hanno utilizzato i tempi di lettura/scrittura per una dimensione di un file tra 512MB e 2GB.

Come si può vedere nelle distinte figure, il rendimento del dispositivio Fibre Channel è abbastanza superiore a le soluzioni software proposti. Questo è logico perché:

- Il tipo di conessione utilizzata tra le macchine el il disco Fibre Channel è di 2Gb/s mentre che la conessione tra il fileserver e le macchine è di 1Gb/s.
- Il disco Fibre Channel è un dispositivo hardware specifico per essere utilizzato come SAN , mentre che il fileserver no.

Nonostante questo, il prezzo dei dispositivi Fibre Channel è abbastanza elevato, mentre che implementare una soluzione basata su iSCSI oppure su GNBD è più affordabile. Inoltre, questi due soluzione sono affidabili –infatti sono tecnologia abbastanza mature– e il rendimento che offrono è anche elevato –in relazione con il suo costo di implementazzione–.

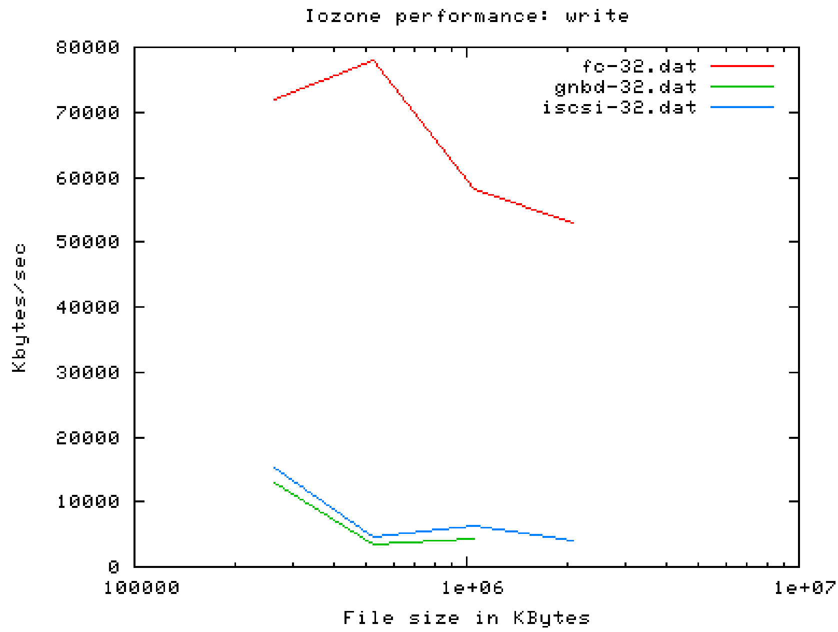



Figura 3.18: Confronto 2D risultati scrittura a 32 bit.

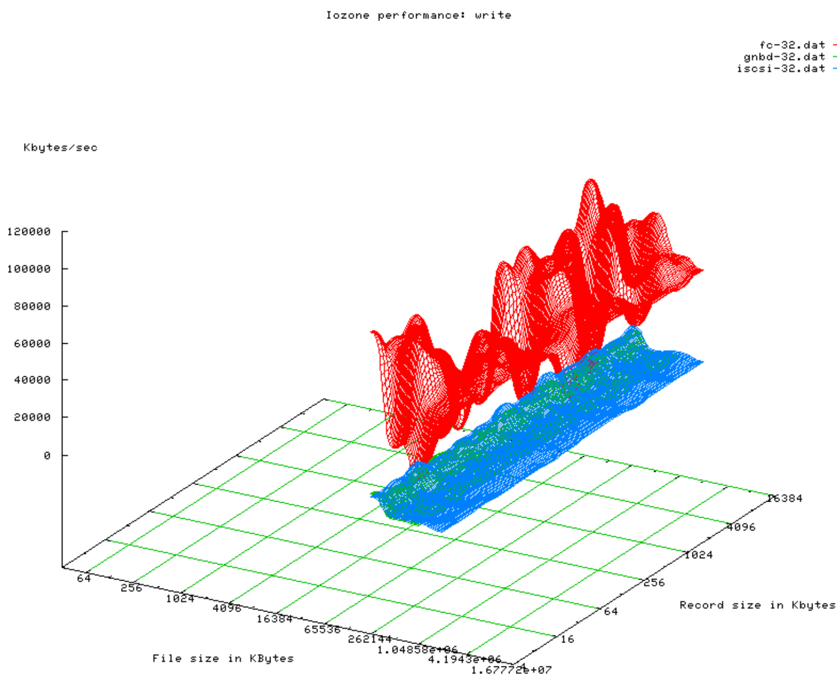



Figura 3.19: Confronto 3D risultati scrittura a 32 bit.

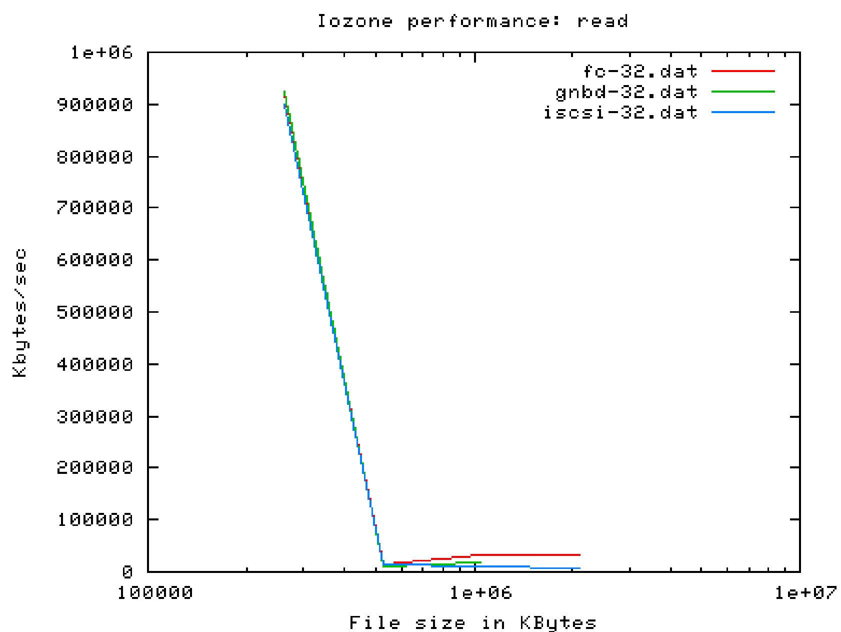



Figura 3.20: Confronto 2D risultati lettura a 32 bit.

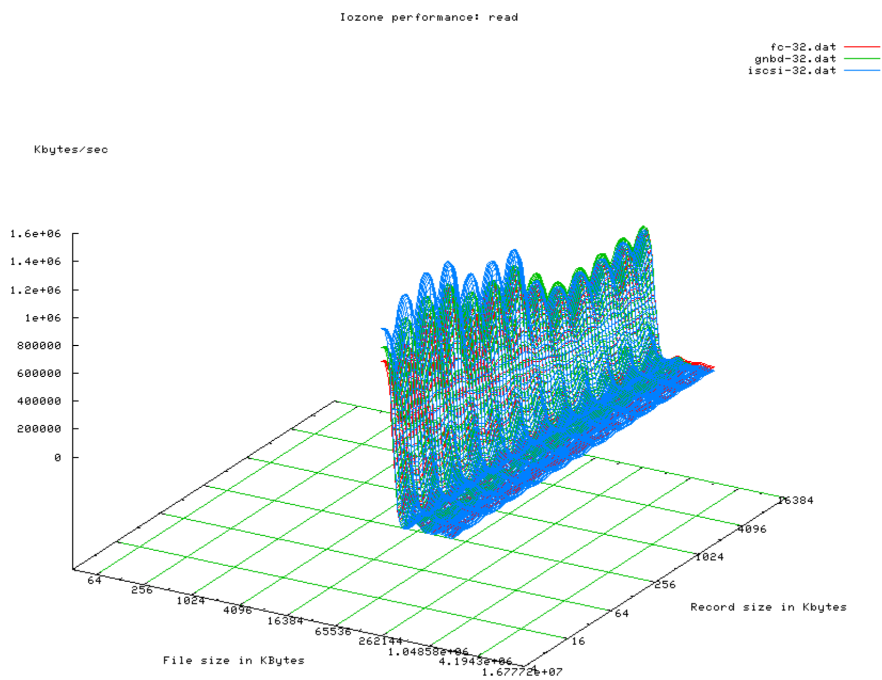



Figura 3.21: Confronto 3D risultati lettura a 32 bit.

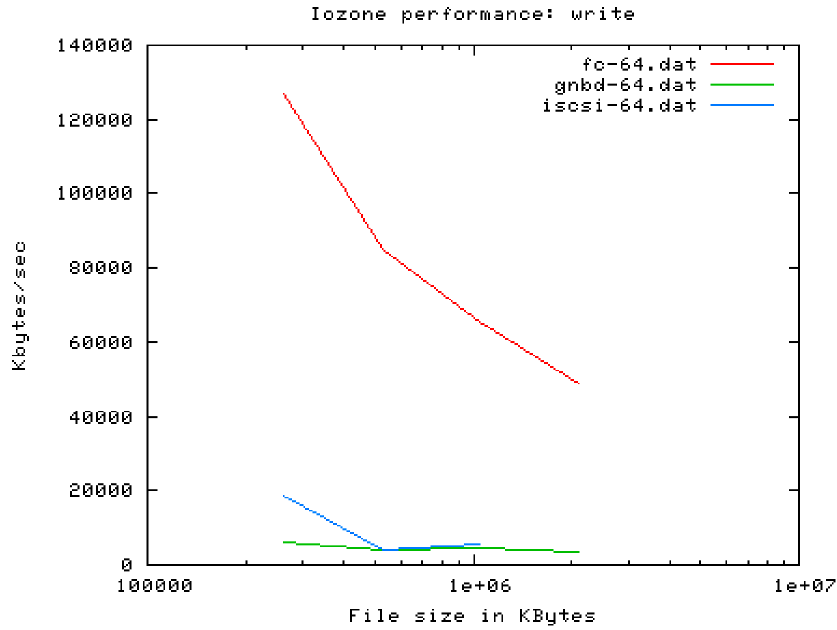



Figura 3.22: Confronto 2D risultati scrittura a 64 bit.

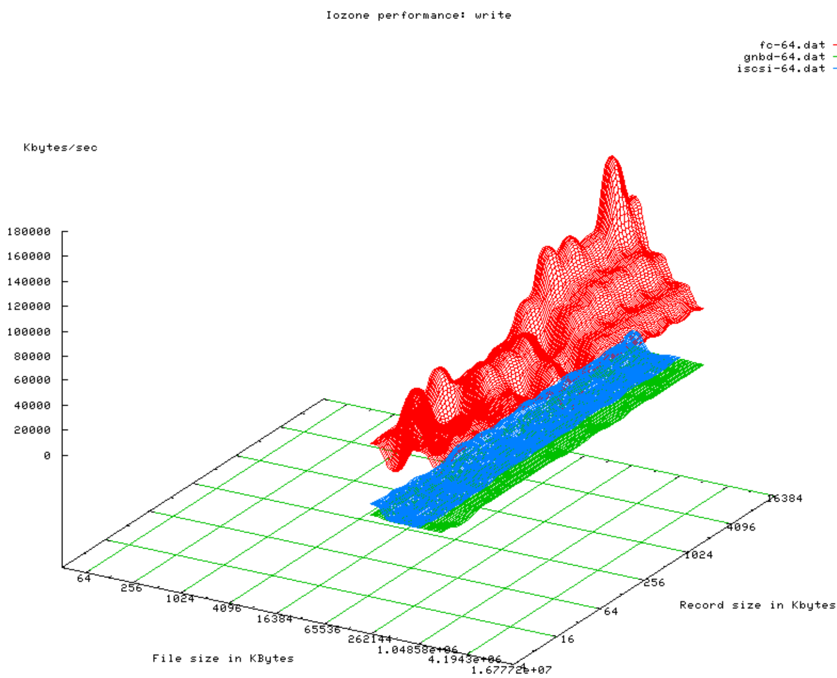



Figura 3.23: Confronto 3D risultati scrittura a 64 bit.

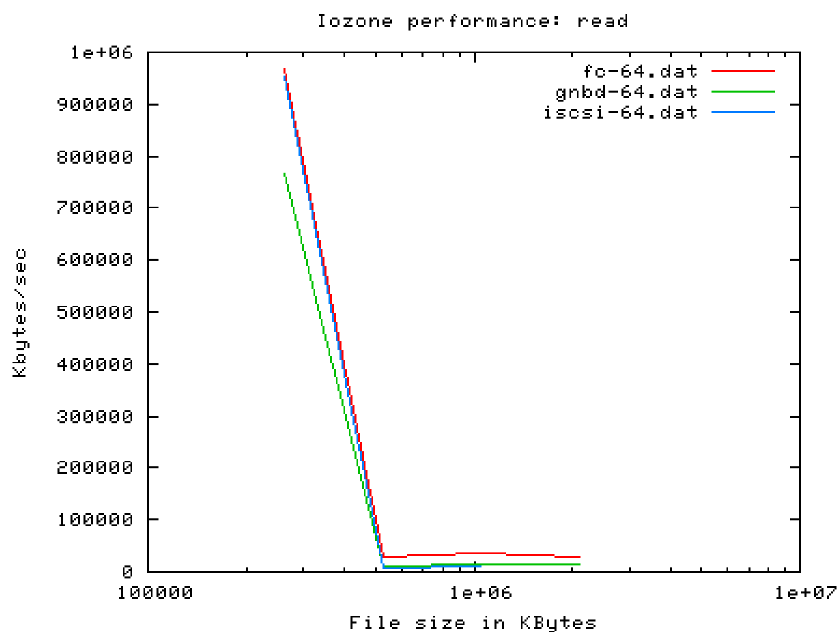



Figura 3.24: Confronto 2D risultati lettura a 64 bit.

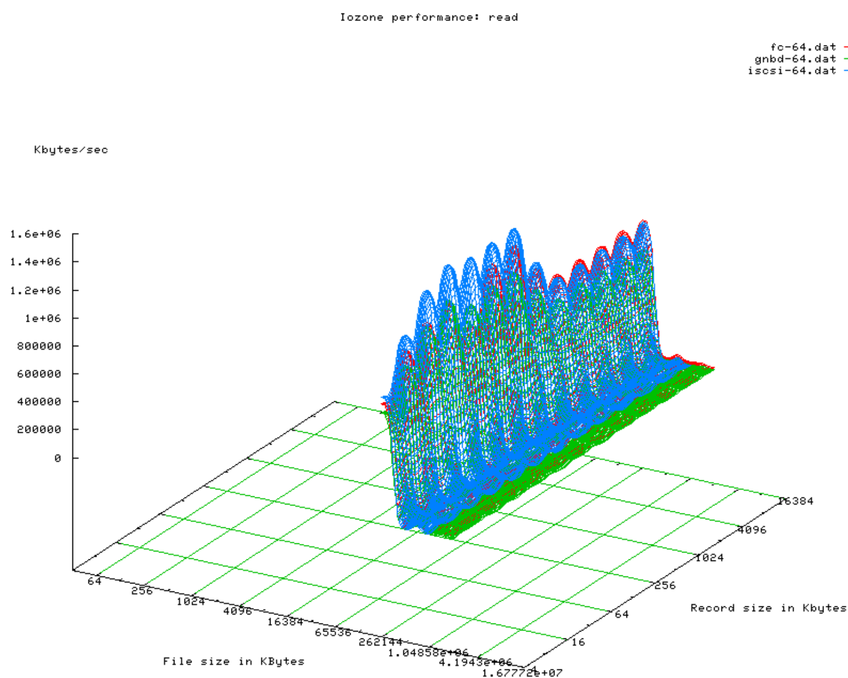



Figura 3.25: Confronto 3D risultati lettura a 64 bit.

## 3.5 Healthcheck

Come servizio di healthcheck[9] si intende un servizio capace di capire lo stato di funzionamento di tutte le macchine –sia virtuali che fisiche– e capace di attuare secondo questo stato, cioè:

- Se la esecuzione di una determinata macchina virtuale non è corretta, deve essere capace di spostarla a un'altra macchina fisica.
- Se una macchina fisica si rompe, deve fare ripartire tutte le macchine virtuali in altre macchine fisiche.
- Inviare informazione su le azione fatte agli amministratori del sito.

In definitiva, questo servizio di healthcheck deve attuare come un supervisore che controlla le macchine di forma automatica. Ci sono distinte possibilità per portare avanti questo sistema.

### 3.5.1 Heartbeat

Lo scopo di Heartbeat è provedere alla suite Linux-HA[10] –alla cui appartene– di un tool di monitoraggio e ricupero di nodi entro un cluster.

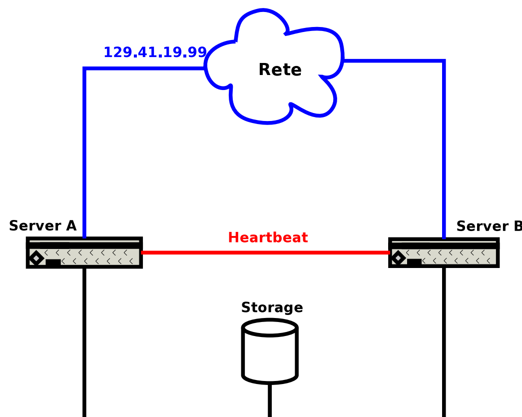



Figura 3.26: Schema di hearbeat.

Nella Figura 3.26 a pagina 40 c'è la struttura di funzionamiento di Heartbeat. Ci sono due server collegati tra una conessione chiamata come el programa stesso: hearbeat. Soltanto uno dei server (A) è collegato alla rete di forma attiva. Se questo server fallisce per qualche causa, la conessione heartbeat è rotta e il server B diventa il server attivo, utilizzando la IP del server A. In questo modo, il tempo di ripristino del servizio è molto piccolo (secondi).

[9] Verificazione della salute, in inglese.
[10] URL: `http://www.linux-ha.org/`

La prima versioni di heartbeat soltanto permetteva la supervisione di due nodi, ma la versione 2 sopporta i nodi senza limiti (sebbene i sviluppatori non lo hanno provato con più di 16 nodi).

### 3.5.2 Nagios

Nagios[11] è una complessa e potente soluzione di monitoraggio di servizi e di rete. È basato in un server che interroga tutti i host e i servizi definiti tramite i distinti plugins disponibili, che danno delle informazione sul servizio a Nagios. Poi, Nagios analizza e memmorizza questa informazzione –accesibile tramite una interfaccia web Figura 3.27 a pagina 41–, attuando in conseguenza a certe regole definite per ogni stato e ogni servizio.

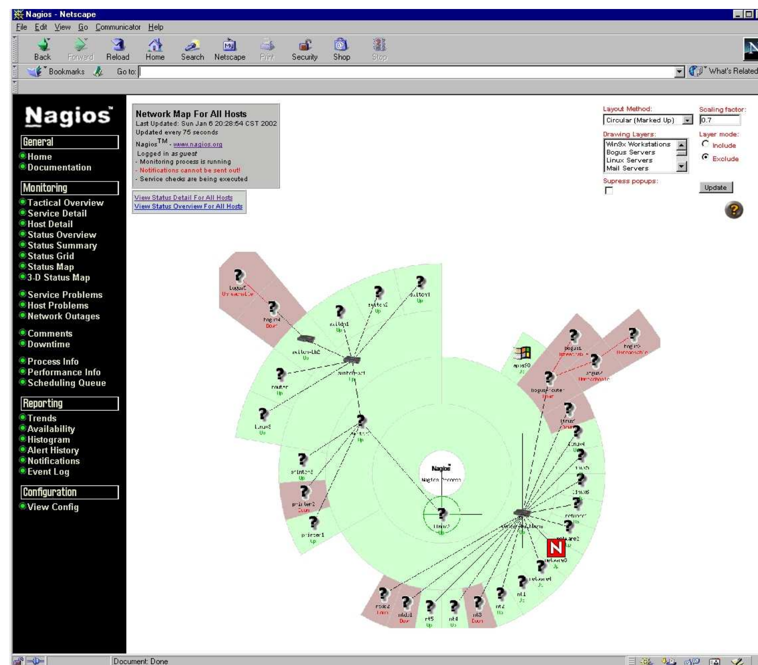



Figura 3.27: Screenshot di nagios.

Nagios è una soluzione interensante, perchè:

- Si possono scribere distinti plugins per distinti servizi.
  - La struttura di Nagios è totalmente adattabile e flessibile.
- Si possono definire le dipendenze fra le macchine e le servizi.
  - Certe macchine virtuali dipendono di una macchina fisica, oppure,

[11] URL: `http://www.nagios.org/`

  - certe servizi dipendo di altri (ad esempio un web server di un DB server).

- Si possono definire le azione a fare a secondo dello stato del servizio/host.

Come conseguenza di tutta questa potenza e addatabilità, Nagios ha come aspetto negativo una curva di apprendimento molto alta al inizio, quindi bisogna un lungo percorso per la persona incaricata della installazione fino che sia una soluzione usabile.

### 3.5.3 Healthcheck propio

In contrapunto a questi due soluzione, che devono essere adatte a la struttura di macchine virtuali, si ha pensato di scrivere un progama in linguaggio C per fare il servizio di healtcheck.

Questo programa consistirebbe in:

**Master** Conosce di forma dinamica la configurazione, e i servizi (e quindi lo suo stato) che girano in ogni macchina, sia fisica che virtuale, conoscendo anche il suo tipo è le dipendenze fra le macchine (che VM gira su ogni macchina fisica).
Questo master fa una comprobazione periodica dello stato memmorizato, interrogando lo slave se è da un certo tempo che non ha ricevuto un informe dello stato dei servizi della macchina. Se ancora non riceve una risposta sattisfattoria –oppure riceve una risposta di fallimento di qualsiasi cosa–, attuerà in conseguenza: spostando le macchine virtuale, riavviando il servizzio oppure avvisando il amministratore del sito.

**Slave** Lo slave invia periodicamente al master –oppure quando è interrogato da lui– la informazione sul nodo e lo stato dei servizi che girano nella sua macchina.
Per prendere lo stato dei servizi, lo slave esegue distinti script che fanno le comprobazione –totalmente customizabili– per ogni servizio. In questo modo lo slave diventa lo stesso per tutte le macchine e bisogna solo scrivere oppure addatare i scripts per ogni servizio.

In questo modo si può implementare una soluzione fatta a posta per il prototipo. Inoltre, durante l'intervallo di sviluppo della soluzione si possono vedere le difficoltà esistenti, quindi questo è un aspetto utile per adottare una nuova soluzione in un futuro (Nagios, Heartbeat, ecc.) e afrontare i problemi con un'altro punto di vista.

# Capitolo 4

# Il prototipo: realizzazione

## 4.1 Sistema di storage

Tra le varie possibilità disponibili nel momento di scrittura della Tesi (Fibre Channel, GNBD, iSCSI) si ha scelto iSCSI per la realizzazione di questo prototipo.

Come si ha visto prima (Capitolo 3), Fibre Channel e la soluzione migliore in quanto riguarda alla velocità, ma il costo di una soluzione Fibre Channel e abbastanza alto e quindi non è così abbordabile.

Inoltre, la scelta di una delle altre due –iSCSI e GNBD– fa possibile testare più in profondità la sua stabilità e scalabilità. Tra questi due si ha scelto iSCSI per sembrare la migliore soluzione (GNBD non esiste come un software indipendente ma è parte di una suite maggiore), la più stabile ed anche per mostrare un rendimento migliore fronte a GNBD (sebbene non così elevato).

## 4.2 Healthcheck

Il sistema di healthcheck utilizzato in questo prototipo è stato scritto in linguaggio C e ha le seguente caratteristiche:

- Il funzionamento è master-slave, cioè, un programma chiamato master controlla tutti gli slave che girano in ogni macchina e inviano informazione sul funzionamento delle macchine e degli servizi.
- Gli slave sono configurabili tramite un file di configurazione.
  - Il master non conosce gli slave finché non riceve la configurazione degli slave tramite UDP.
- La comparazione dei distinti servizi viene fatta tramite scripts generici, totalmente personalizzabili.

- Il master è quello che controlla:
  - Se i servizi che girano su una macchina virtuale funzionano correttamente.
  - Se ogni macchina (sia fisica che virtuale) funziona correttamente. Si una macchina fisica si è rotta, fa ripartire le macchine virtuali in un'altra macchina fisica.

Durante la realizzazione di questo programma di healthcheck si hanno visto distinti problemi nella implementazione. Di questo de delle possibili prospettive future su questo programma si parlerà nel Capitolo 5.

### 4.2.1 Master

Il master riceve tramite messaggi UDP informazione sui distinti slave che devono essere controllati (dunque al inizio della sua esecuzione non conosce nessun slave). Questa informazione è aggiornata in tempo reale con gli spostamenti delle macchine, lo stato dei servizi, ecc.

Il master guarda periodicamente gli stati degli macchine presenti:

- Se una macchina virtuale non risponde, la riavvia.
- Se una macchina fisica non rispondi, riavvia le sue macchine virtuali nelle altre macchine fisiche disponibili.

Inoltre, il suo comportamento con gli servizi è il seguente:

- Se un servizio non funziona, prova di riavviarlo.
- Se ancora non funziona, lo ferma.

Non tutti i messaggi che invia lo slave stanno implementati e riconosciuti nel master, ma si ha fatto in questo modo per favorire una futura dilazioni delle funzionalità (e viceversa).

#### 4.2.1.1 Messaggi riconosciuti

```
ID;<nome>;<vm>;<host>;<critical>;<servizio_1>;...;<servizio_n>
```

**`<nome>`** Il nome della macchina.

**`<vm>`** 0 se è una macchina fisica, 1 se è una macchina virtuale.

**`<host>`** Se è una macchina virtuale questo campo ha la IP della macchina virtuale che la esegue.

**`<critical>`** 0 se è una macchina critica, se non lo è (non implementato).

**<servizio_1>;...;<servizio_n>** Lista di servizi disponibili.

**SD;<nome_servizio>;<stato>**

**<nome_servizio>** Il nome del servizio controllato.

**<stato>** Lo stato del servizio controllato.

#### 4.2.1.2 Messaggi inviati

**receivedid** Messaggio inviato quando riceve la configurazione di una macchina.

**autorestart** Messaggio inviato quando esiste qualche errore riparabile nella configurazione di una macchina.

**errorvm** Messaggio inviato quando una macchina virtuale non tiene una macchina host.

**errorsv** Messaggio inviato quando un servizio non esiste e si riceve informazione su questo servizio.

**collecteddata** Messaggio inviato per chiedere allo slave la informazione disponibili sui servizi.

**AC;VM;DESTROY;<nome_nodo>** Messaggio inviato per distruggere la macchina `<nome_nodo>`.

**AC;VM;CREATE;<nome_nodo>.cfg** Messaggio inviato per creare la macchina con file di configurazione `<nome_nodo>.cfg`.

**AC;SV;<servizio>;stop** Messaggio inviato per fermare il servizio <servizio>.

**AC;SV;<servizio>;restart** Messaggio inviato per riavviare il servizio <servizio>.

### 4.2.2 Slave

Lo slave e configurabile tramite il file mostrato in C.2.4. Invia la sua configurazione al master tramite un pacchetto UDP e poi comincia a comprovare lo stato dei servizi tramite gli scripts corrispondenti.

Anche come il master, non tutti i messaggi che invia lo slave stanno implementati e riconosciuti nel master e viceversa.

#### 4.2.2.1 Messaggi riconosciuti

**autorestart** Messaggio per riavviare lo slave.

**receivedid** Messaggio del master per comunicare che ha ricevuto la ID.

**collecteddata** Messaggio per chiedere gli stati degli distinti scripts.

**status** Messaggio per chiedere lo stato della macchina.

**sendid** Messaggio per chiedere allo slave di inviare di nuovo la sua ID.

**errorvm** Messaggio che riavvia lo slave perché esiste un errore nella configurazione quando è una VM.

**AC;REBOOT** Messaggio per riavviare la macchina.

**AC;SV;<servizio>;[STOP|START|RESTART]** Messaggio per fermare, lanciare o riavviare un servizio.

Messaggi soltanto per le macchine fisiche:

**AC;VM;[SHUTDOWN|REBOOT|DESTROY];<nome_nodo>** Messaggio per fermare, riavviare o distruggere la macchina virtuale chiamata `<nome_nodo>`.

**AC;VM;CREATE;<file_configurazione>** Messaggio per creare una macchina virtuale secondo il file `<file_configurazione>`.

**AC;VM;MIGRATE;<nome_nodo> <nome_nuovo_host>** Messaggio per migrare la macchina `<nome_nodo>` a `<nome_nuovo_host>`.

**AC;VM;LMIGRATE;<nome_nodo> <nome_nuovo_host>** Idem per una live migration.

#### 4.2.2.2 Messaggi inviati

Lo slave invia al master la uscita di tutti le azioni indicati per i messaggi che cominciano per AC. Anche invia i messaggi presenti in 4.2.1.1.

#### 4.2.2.3 Gli scripts

Gli scripts per comprovare lo stato di un servizio devono essere scritti per l'utente. Possono essere qualsiasi file eseguibile e devono soddisfare tre condizioni:

- Il valore di ritorno se il servizio funziona deve essere 0. Se non funziona serve qualsiasi altro valore.
- La azione per difetto deve essere la verifica del servizio.
- Deve accettare come parametri start, stop e restart; per lanciare, fermare e riavviare il servizio rispettivamente.

## 4.3 Il prototipo

Il prototipo creato –come prova di concetto– è stato basato nel funzionamento di 6 macchine virtuali su due macchine fisiche, tutti controllati tra il servizio di Healthcheck (a seconda della struttura mostrata nella Figura 3.1 a pagina 22) e la simulazione dei fallimenti delle macchine (sia fisica che virtuali).

Com'è normale e come si aspettaba, il funzionamiento dei servizi Grid su macchine virtuali non ha raportato problemi considerabili.

Il servizio di Healthcheck scritto ha fatto la sua funzione nei casi previsti, ma si hanno visto possibili problemi in questa struttura e anche possibili perfezionamenti di tutto il prototipo, dei cui si parlerà nel Capitolo 5.

Per ultimo si ha visto che la struttura proposta è solida, stabile, e permette il ripristino dei servizi e delle macchine di forma automatica in secondi (oppure pochi minuti) . Dunque, con la adozione di un sistema de healthcheck più maturo e che controlla tutte gli scenari possibili potrà diventare una potentissima soluzione per la alta affidabilità.

# Capitolo 5

# Prospettive future

**Implementazione dell prototipo su larga scala** Bisogna guardare la scalabilità del prototipo, i problemi presenti, la variazione del rendimento in una implementazione a grande scala, ecc.

**Possibilità di utilizzare Nagios come servizio di Healthcheck** Durante lo sviluppo della soluzione di Healthcheck se hanno trovato parecchie problemi che rendono il disegno e la implementazione di questo istrumento abbastanza difficile:

- Dipendenze fra le macchine.
- Dipendenze fra i servizi.
- Bisogna scrivere un script per ogni servizio che si vuole controllare.
- Le migrazione delle macchine è una materia complicata e delicata (riallocazione di memoria, algoritmo di *load-balancing*, ecc)

Nagios invece è una soluzione matura e avanzata, pensata per monitorare e controllare grandi quantità di nodi, quindi lo sviluppo di un plugin per essere utilizzato per monitorare sia le macchine fisiche che virtuali potrà diventare una soluzione potente. Inoltre, ci sono già sviluppati plugin per qualsiasi servizio installabile in una macchina GNU/Linux.

**Load-Balancing** Una caratteristica importante a implementare nel servizio di healthcheck sarebbe un algoritmo di *load-balancing* in modo che le macchine virtuali possano essere spostati a un'altra a seconda del suo carico.

Questa caratteristica esiste nel *roadmap*[1] di Xen e anche la Red Hat ultimamente ha cominciato lo sviluppo del *load-balancing* in un cluster di macchine Xen, ma mentre che non sia totalmente sviluppata, e interessante la sua implementazione tramite il healthcheck.

[1] Xen Roadmap `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/roadmap.html`

**Ottimizzare l'uso del filesystem** In questo punto ci sono varie possibilità per essere utilizzati.

**Condivisione di una parte del filesystem read-only di forma remota e ogni nodo con il resto del suo filesystem di forma remota.** In questo modo tutti i nodi similari (per esempio tutti i Worker Node (WN) ) condividono i pezzi del filesystem statici, ottimizzando il uso del spazio disco.

**Condivisione di una parte del filesystem read-only di forma remota e ogni nodo con il resto del suo filesystem di forma locale** Questa soluzione rapporta lo stesso vantaggio che la soluzione previa, ma anche aggiunge il vantaggio di utilizzare il disco locale per fare tutte le operazione di scrittura, riducendo il carico nella rete e quindi rapportando un rendimento migliore.

Un grande svantaggio è che se si fa una migrazione di una macchine, lo spazio disco locale si perde (e quindo tutto il lavoro fatto).

**Uso delle soluzione anteriori con LVM** Anche l'uso di LVM è interessante, perché si può sincronizzare ogni certo tempo lo spazio disco locale con un filesystem remoto, permettendo in questo punto la migrazione delle macchine.

**Integrazione di Xen con sistemi di installazione automatizzati** In concreto con quello utilizzato per il LCG , cioè LCG-yaim e con IG-yaim (utilizzato per l'INFN-Grid).

In questo modo aggiungere un nuovo nodo si può fare di forma semiautomatica e con la sicurezza che la configurazione sarà sempre la corretta.

# Ringraziamenti

Al Prof. Leonello Servoli per aiutarmi con la burocrazia quando sono proprio arrivato, per avermi permetto lavorare con lui e per la sua pazienza con il mio italiano. A Mirko Mariotti e Igor Neri per l'aiuto tecnico, i suoio sugerimenti, per tutti i pranzi insieme e anche per aiutarmi a pulire il mio italiano. A Ivan Grasso per il suo supporto (ed anche un'altra volta per aiutarmi con l'italiano!). A tutti i miei amici (perugini, spagnoli ed spagnoli a Perugia). A la mia famiglia. A tutti quanti che hanno creduto in me. Ed alla fine, a María, per avermi dato forza per venire a Italia e per aspettarmi a 2000Km fin adesso.

# Appendice A

# Installazione e uso di Xen

## A.1 Installazione di Xen

Si è scelto Xen 3.0.2 perché è l'ultima versione stabile disponibile al momento dello sviluppo del prototipo. Per l'installazione basta seguire le istruzioni disponibili nella documentazione di Xen[7]. Bisogna fare attenzione ai prerequisiti presenti nella documentazione, perché potrebbero nascere diversi problemi se non si soddisfano.

Nel momento di scrittura della tesi, la versione di `pygrub` che viene distribuita con Xen 3.0.2 è errata e c'è bisogno di installare quella distribuita con la versione instabile di Xen. Per fare questo, si può scaricare la versione instabile dal sito web `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/downloads.html` e procedere all'installazione:

```
# tar zxfv xen-unstable-src.tgz
# cd xen-unstable/tools/pygrub/
# make build
# make install
```

A questo punto, l'installazione di Xen dovrebbe essere pronta. Una volta riavviata la macchina, `xend` può essere lanciato:

```
# xend start
```

Se ci sono problemi o errori, consultare la documentazione di Xen.

## A.2 Configurazione di Xen

### A.2.1 Configurazione di xend

La configurazione del daemon xend viene nel file /etc/xen/xend-config.spx. La configurazione per difetto è quasi completamente valida ma bisogna cambiare una righa nel file per permettere la migrazione delle macchine virtuali:

```
#(xend-relocation-server no)
```

per

```
(xend-relocation-server yes)
```

### A.2.2 Configurazione dei macchine virtuali

Per ogni macchina virtuale bisogna un file di configurazione con la seguente struttura:

```
kernel = "/boot/kernel-2.6.12.6-xenU.64"
memory = 512
name = "xen-test"
disk = ['phys:/dev/sdb1,sda1,w']
root = "/dev/sda1 ro"
vif = ['mac=00:00:00:00:02:07']
dhcp = "dhcp"
```

**kernel** Il path al kernel per utilizzare.

**memory** La memmoria RAM per essere utilizzata.

**name** Il nome della macchina (se si usa DHCP questo nombre sarà cambiato per quello fornito da DHCP).

**disk** Il block device in cui stà il filesystem della macchina virtuale.

**root** Dove sarà la montata la partizione root.

**vif** Il MAC address della macchina (qui, se non si usa DHCP si può anche specificare l'indirizzo IP).

**dhcp** Se questa variabile ha come valore "dhcp" questo servizio verrà utilizato.

### A.2.3 pyGRUB

Se invece di utilizzare un kernel esterno alle macchine virtuale si vuole utilizzare un kernel entro la immagine della macchina, se deve utilizzare `pygrub`. Quindi, nel file di configurazione bisogna cambiare

```
kernel = "/boot/kernel-2.6.12.6-xenU.64"
```

per

```
bootloader = "/usr/bin/pygrub"
```

Nella immagine deve esistere il file /boot/grub.conf con una configurazione valida per il kernel installato nella macchina. A questo punto, ogni volta que si avvie una nuova macchina virtuale sarà un menu in cui scegliere il kernel a utilizzare.

```
   pyGRUB  version 0.3
??????????????????????????????????????????????????????????????????????????
? Linux 32bit kernel 2.6.12.6-xen                                        ?
? Linux 32bit kernel 2.6.16-xen                                          ?
?                                                                        ?
?                                                                        ?
?                                                                        ?
?                                                                        ?
?                                                                        ?
?                                                                        ?
??????????????????????????????????????????????????????????????????????????
    Use the ^ and ? keys to select which entry is highlighted.
    Press enter to boot the selected OS. 'e' to edit the
    commands before booting, 'a' to modify the kernel arguments
    before booting, or 'c' for a command line.
```

Figura A.1: Menu di pyGRUB

# A.3 Uso di Xen

## A.3.1 Lanciamento di Xen

Xen funziona tramite un daemon chiamato xend (Vedere 2.4.4). Per lanciare questo daemon basta eseguire

```
# xend start
```

## A.3.2 Il tool xm

Xm e il tool presente nella distribuzione di Xen per controllare le macchine virtuali. Xm può essere utilizzato per creare, distruggere, mettere in pausa, migrare, reassegnare memoria, ecc. Il uso è sempre lo stesso:

```
# xm <ordine> [argomenti]
```

I comandi più utilizzati sono:

**xm create <config_file>** Crea una macchina virtuale secondo il file di configurazzione `<config_file>`.

**xm list** Lista le macchine virtuali attive nel seguente formato:

```
# xm list
Name                                    ID Mem(MiB) VCPUs State  Time(s)
Alvaroxentest206                        18      512     1 -b----   343.5
Alvaroxentest207                        19      512     1 -b----   363.1
Domain-0                                 0      485     2 r----- 17502.3
```

**xm console <dom_id>** Si collega come una consola alla macchina `<dom_id>`. `<dom_id>` può essere il nome della macchina oppure il ID presente quando si fà un `xm list`.

**xm shutdown <dom_id>** Fa un shutdown della macchina virtuale <dom_id>.

**xm destroy <dom_id>** Distrugge la macchina <dom_id>.

**xm reboot <dom_id>** Riavvia la macchina <dom_id>.

**xm migrate <dom_id> <host>** Migra la macchina <dom_id> al host <host> (che ovviamente deve eseguire Xen).

**xm_migrate --live <dom_id> <host>** Migra la macchina <dom_id>, senza metterla in pausa, al host <host>.

**...**

## A.4 Creazione di una VM SL4

Di seguito si descrivono le fasi per creare un'immagine funzionante di SL4 sotto Xen 3.0.2. In questo esempio l'installazione viene fatta in `/dev/sdb1`.

### A.4.1 Installazione di SL4

Si è scelto di fare un'installazione minimale di SL4 in un block device accessibile da una macchina funzionante con GNU/Linux (`/dev/sdb1`). Perciò si sono scelti i pacchetti indispensabili per fare funzionare il sistema e la rete. A questo punto, è possibile installare qualsiasi sistema su questa installazione minimale via `yum` oppure `apt-get`. È importante far notare che non si deve installare né SE-Linux né GRUB (oppure LILO).

Una volta che si è fatta questa installazione minimale, avviando la macchina con il sistema GNU/Linux originale, è necessario accedere alla partizione con SL4 e copiare i contenuti di `/dev` nella partizione con SL4:

```
# mount /dev/sdb1 /mnt/test
# cp -a /dev/* /mnt/test/dev/
```

È anche necessario disattivare udev perché quando la macchina sarà virtualizzata non dovrà accedere a nessun device, perché la gestione dei device viene fatta dal dom0:

```
# mv /mnt/test/sbin/start_udev /mnt/test/sbin/start_udev.no
```

### A.4.2 Compilazione del kernel 2.6.16-xen

È necessario fare una compilazione del kernel 2.6.16-xen per la nuova macchina virtuale:

```
# cd /usr/src/linux-2.6.16-xen
# make menuconfig
# make
# cp vmlinuz /boot/kernel-2.6.16-xenU
# cp .config /boot/config-2.6.16-xenU
```

### A.4.3 Creazione e prova delle macchine virtuali

Per provare se il nuovo kernel funziona si deve creare un file di configurazione, ad esempio `test.cfg`:

```
kernel = "/boot/kernel-2.6.12.6-xenU.64"
memory = 512
name = "xen-test"
disk = ['phys:/dev/sdb1,sda1,w']
root = "/dev/sda1 ro"
vif = ['mac=00:00:00:00:02:07']
dhcp = "dhcp"
```

Il server DHCP deve essere configurato per assegnare un indirizzo IP alla macchina virtuale. A questo punto la VM potrebbe essere lanciata:

```
# xm create -c test.cfg
```

Se non ci sono errori, l'immagine creata è pronta per essere usata. Si deve soltanto copiare il nuovo kernel funzionante in `/boot/` nella macchina virtuale e creare un file di configurazione di grub valido in `/boot/grub.conf` (sempre nella macchina virtuale) per usare `pygrub` invece di avere un kernel esterno alla macchina virtuale. Sarebbe anche opportuno aggiornare tutti i pacchetti della macchina virtuali ed anche installare `gcc`.

# Appendice B

# Uso di block device via rete

I block device via rete permettono l'esportazioni di device attraverso una rete Ethernet. In questo appendice si afrontarà la installazione delle due soluzione utilizzati nella tesi: Internet SCSI (iSCSI) e Global Network Block Device (GNBD) .

## B.1 iSCSI

### B.1.1 Introduzzione

iSCSI ha una struttura client-server. Il cliente viene chiamato *initiator* –cioè, quello que comincia una petizione di una risorsa– mentre il server viene chiamato *target*.

### B.1.2 Installazione di un target

#### B.1.2.1 Installazione

La implementazione utilizata viene chiamata *"The iSCSI Enterprise Target"* `http://iscsitarget.sourceforge.net/`. Una volta scaricati i sorgenti si possono compilare e installare tramite i comandi:

```
# make KERNELSRC=/usr/src/linux
# make KERNELSRC=/usr/src/linux install
```

dove linux è un link simbolico che punta al kernel sul quale si vuole compilare iSCSI.

#### B.1.2.2 Configurazione

La configurazione viene fatta tramite il file `/etc/ietd.conf` che ha una o più delle seguenti strutture:

```
Target iqn.1997-01.it.infn:pg.na48fs3.xentest206
        # Lun definition
```

```
Lun 0 Path=/data16/images/206.img,Type=fileio
Alias xentest206
```

In questa configurazione si define un dispositivo per essere esportato, con l'*iSCSI Qualified Name* `iqn.1997-01.it.infn:pg.na48fs3.xentest206` [14]. Il block device esportato sarà un file (con una immagine entro lui) chiamato `/data16/images/206.img`.

Per più informazione su la configurazione e definizione dei block device a esportare, si può guardare la bibliografia [15].

#### B.1.2.3 Lanciamento del servizio

Per lanciare il servicio si deve eseguire il comando:

```
/etc/init.d/iscsi-target start
```

### B.1.3 Installazione di un initiator

La distribuzione utilizata è *"core-iSCSI"*, consistente in due paccheti: Un modulo per il kernel è i tools per utilizare iSCSI. Quesi due paccheti si chiamano core-icsi-tools `http://www.kernel.org/pub/linux/utils/storage/iscsi/` e core-iscsi `http://www.kernel.org/pub/linux/kernel/people/nab/iscsi-initiator-core/`.

#### B.1.3.1 Installazione di core-iscsi

La installazione viene fatta tramite i comandi:

```
# make initiator KERNEL_DIR=/usr/src/linux
# make install
```

dove linux è un link simbolico che punta al kernel sul quale si vuole compilare iSCSI.

#### B.1.3.2 Installazione di core-iscsi-tools

La installazione viene fatta tramite il comando:

```
# make install
```

#### B.1.3.3 Configurazione

Ci sono vari file per essere configurati, ma in questa spiegazione soltanto si parlerà dei file necesari per fare andare avanti il *initiator* con la configurazione anteriormente mostrata. Per pì informazione bisogna guardara la bibliografia [17].

**/etc/initiatorname.iscsi** Sarà il *iSCSI Qualified Name* della macchina.

**/etc/sysconfig/initiator** In questo file ci sono i distinti *target* a cui la macchina si collegarà. Per la configurazione previa del *target* sarà (in una unica righa):

```
CHANNEL="0 1 eth0 192.168.254.54 3260 0 AuthMe-
thod=None;MaxRecvDataSegmentLength=8192 nopout_timeout=5
                         iqn.1997-01.it.infn:pg.na48fs3.xentest206"
```

#### B.1.3.4 Lanciamento del servizio

Per utilizare i dispositivi tramite iSCSI soltanto manca eseguire il commando:

```
/etc/init.d/initiator start
```

Tramite il commando

```
/etc/init.d/initiator status
```

si ottiente informazione delle dispositivi (tra altri, il dispositivo creato):

```
# /etc/init.d/initiator status
——————[iSCSI Session Info for iSCSI Channel 0]——————
TargetName: iqn.1997-01.it.infn:pg.na48fs3.xentest206
TargetAlias:
PyX Session ID: 9  ISID: 0x80 33 94 32 00 00  TSIH: 4864
Cmds in Session Pool: 32  Session State: INIT_SESS_LOGGED_IN
———————[iSCSI Session Values]————————
   CmdSN   :  ExpCmdSN  :  MaxCmdSN  :    ITT    :    TTT
  0x00098669   0x00098669   0x00098689   0x000a8b7e   0x0006dd91
————————-[iSCSI Connections]————————-
CID: 0  Connection State: INIT_CONN_LOGGED_IN
       Address 192.168.254.54:3260,1 TCP  ExpStatSN: 0x000a8b7d
————————[SCSI Info for iSCSI Channel 0]————————-
SCSI Host No: 8  SCSI-II Host TCQ Count: 128
Logical Unit TCQ Depth: 64  SGTableSize: 32  MaxSectors: 256
iSCSI Logical Unit Number: 0  Status: ONLINE -> READ/WRITE
        DISK: sdd        SCSI BUS Location: 0/0/0  Sector Size: 512
       Active Tasks: 11  Total Tasks: 1311950  Total Bytes: 69679982272k
```

## B.2 GNBD

### B.2.1 Introduzione

GNBD è suddiviso in due componenti, una parte server che si occupa dell'esportazione dei device locali ed una parte client che si occupa di importare i device da una specifica macchina.

### B.2.2 Installazione di GNBD

GNBD fa parte di una suite più grande di software RedHat, denominata cluster. Per l'installazione si devono scaricare i sorgenti da `ftp://sources.redhat.com/pub/cluster/releases` ed in seguito compilare attraverso i comandi:

```
# ./configure --kernel_src=/usr/src/linux
# make install
```

dove linux è un link simbolico che punta al kernel sul quale si vuole compilare GNBD.

### B.2.3 Esportazione e Importazione

Sulla macchina server una volta compilato GNBD bisogna far partire il demone di servizio con il comando

```
# gnbd_serv
```

ed esportare con l'istruzione –che ovviamente si può utilizzare più volte per esportare più di un device–

```
# gnbd_export -c -e <nome_univoco_device> -d <nome_block_device>
```

Per quanto riguarda le macchine client bisogna prima caricare il modulo del kernel con il comando

```
# modprobe gnbd
```

e poi importare con

```
# gnbd_import -i <server_gnbd>
```

Una volta fatto questo, i devices imporati saranno disponibili in

```
/dev/gnbd/
```

# Appendice C

# Healthcheck

## C.1 Uso

Il uso di questo programma e semplice. Basta modificare il file di configurazione, eseguire il master in un calcolatore ed eseguire un slave per ogni macchina che si vuole controllare.

## C.2 Sorgenti

### C.2.1 master.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <malloc.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include "control.h"
#include "list.h"
int client_status[NUMCLIENTS];
char * client_ip[NUMCLIENTS];
struct service_list {
    char * service;
    int status;
    time_t heartbeat;
    int count;
    struct list_head list;
};
enum critical {no=0,yes=1};
struct node_list {
    char * name;
```

```
    char * ip;
    char * host;
    int control;
    enum critical critical;
    time_t heartbeat;
    int count;
    int status;
    int isVM;
    struct service_list services;
    struct node_list * vms;
    struct list_head list;
};
struct node_list pms;
/* mutex on the client_status array */
pthread_mutex_t cs_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t list_mutex = PTHREAD_MUTEX_INITIALIZER;
/* mutex on logging */
pthread_mutex_t log_mutex = PTHREAD_MUTEX_INITIALIZER;
/* mutex on the outgoing UDP */
pthread_mutex_t udp_out_mutex =  PTHREAD_MUTEX_INITIALIZER;
/* This is the message logging function */
void do_log(char * message) {
    FILE * logfile;
    time_t current_time;
    char *line;
    line =(char *) calloc (30+strlen(message),sizeof(char));
    current_time=time(NULL);
    strncpy(line,ctime(&current_time),24);
    line[24]='\0';
    strcat(line," ");
    strcat(line,message);
    pthread_mutex_lock (&log_mutex);
    fprintf(stderr,"%s\n",line);
    pthread_mutex_unlock (&log_mutex);
}
/* Return any node with that IP */
struct node_list * search_ip(char ip[15]) {
    struct node_list *vm_tmp, *pm_tmp;
    struct list_head *vm_pos, *pm_pos;
    list_for_each(pm_pos, &pms.list){
        pm_tmp=list_entry(pm_pos, struct node_list, list);
        if(!strcmp(pm_tmp->ip,ip))
            return(pm_tmp);
        else
            list_for_each(vm_pos, &pm_tmp->vms->list) {
                vm_tmp= list_entry(vm_pos, struct node_list, list);
                if(!strcmp(vm_tmp->ip,ip))
                    return(vm_tmp);
            }
    }
    return(NULL);
}
/* Return the PM which has less VMs */
struct node_list * search_less_vms(char * ip) {
    struct node_list * node_tmp, * im_the_node;
    struct list_head * pos;
    im_the_node=NULL;
    list_for_each(pos, &pms.list) {
        node_tmp=list_entry(pos,struct node_list, list);
        if(im_the_node==NULL ||
           im_the_node->control < node_tmp->control && strcmp(node_tmp->ip,ip)) {
            im_the_node=list_entry(pos,struct node_list, list);
```

```
        }
    }
    return(im_the_node);
}
/* Return the *pos of a PM w/ such IP */
struct list_head * search_pos_pms_ip(char ip[15]) {
    struct node_list *tmp;
    struct list_head *pos;
    list_for_each(pos, &pms.list){
        tmp= list_entry(pos, struct node_list, list);
        if(!strcmp(tmp->ip,ip))
            return(pos);
    }
    return(NULL);
}
/* Return the PM with such IP*/
struct node_list* search_pms_ip(char ip[15]) {
    struct node_list *tmp;
    struct list_head *pos;
    list_for_each(pos, &pms.list){
        tmp= list_entry(pos, struct node_list, list);
        if(!strcmp(tmp->ip,ip))
            return(tmp);
    }
    return(NULL);
}
/* Return the VM with such ip */
struct node_list* search_vms_ip(char ip[15]) {
    struct node_list *vm_tmp, *pm_tmp;
    struct list_head *vm_pos, *pm_pos;

    list_for_each(pm_pos, &pms.list){
        pm_tmp=list_entry(pm_pos, struct node_list, list);
        list_for_each(vm_pos, &pm_tmp->vms->list) {
            vm_tmp= list_entry(vm_pos, struct node_list, list);
            if(!strcmp(vm_tmp->ip,ip))
                return(vm_tmp);
        }
    }
    return(NULL);
}
/* Delete the VM w/ such IP */
int delete_vm_ip(char ip[15]) {
    struct node_list *vm_tmp, *pm_tmp;
    struct list_head *vm_pos, *pm_pos;
    list_for_each(pm_pos, &pms.list){
        pm_tmp=list_entry(pm_pos, struct node_list, list);
        list_for_each(vm_pos, &pm_tmp->vms->list) {
            vm_tmp= list_entry(vm_pos, struct node_list, list);
            if(!strcmp(vm_tmp->ip,ip)) {
                list_del(vm_pos);
                return(0);
            }
        }
    }
    return(-1);
}
/* Return a pointer to a service of that node */
struct service_list * search_service_ip(char* service,struct node_list* node) {
    struct list_head *pos;
    struct service_list *tmp;
    list_for_each(pos,&node->services.list) {
```

```
        tmp=list_entry(pos,struct service_list,list);
        if(!strcmp(tmp->service,service))
            return(tmp);
    }
    return(NULL);
}
/* This function allow to send UDP to a slave identified by slave_id */
void send_udp(char * message, char ip[15]) {
    int sd, rc;
    struct sockaddr_in cliAddr, remoteServAddr;
    struct hostent *h;
    #ifdef DEBUG
    char *log_string;
    #endif
    pthread_mutex_lock (&udp_out_mutex);
    h = gethostbyname(ip);
    remoteServAddr.sin_family = h->h_addrtype;
    memcpy((char *) &remoteServAddr.sin_addr.s_addr, h->h_addr_list[0], h->h_length);
    remoteServAddr.sin_port = htons(SLAVE_PORT);
    sd = socket(AF_INET,SOCK_DGRAM,0);
    if(sd<0) {
        do_log("Udp gateway: Cannot open socket.");
        return;
    }

    cliAddr.sin_family = AF_INET;
    cliAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    cliAddr.sin_port = htons(0);

    rc = bind(sd, (struct sockaddr *) &cliAddr, sizeof(cliAddr));
    if(rc<0) {
        do_log("Udp gateway: Cannot bind port.");
        return;
    }
    rc = sendto(sd, message, strlen(message)+1, 0,
           (struct sockaddr *) &remoteServAddr, sizeof(remoteServAddr));
    if(rc<0) {
        do_log("Udp gateway: Cannot send data.");
        close(sd);
        return;
    }
    #ifdef DEBUG
    else {
        if (VERBOSITY>4) {
            log_string = (char *) calloc (40+strlen(message),sizeof(char));
            sprintf(log_string,"Udp gateway: Sending message (%s)",message);
            do_log(log_string);
        }
    }
    #endif
    close (sd);
    pthread_mutex_unlock (&udp_out_mutex);
    return;
}
/* Parsing the received message */
void parse(char * msg,char slaveIP[15]) {
    char * aux, * token;
    char log_string[MAX_LOG_MSG];
    struct node_list *node_tmp, *node_host_tmp;
    struct service_list *service_tmp;
    struct list_head *pos;
    token=(char *) calloc(strlen(msg),sizeof(char));
```

```
aux=(char *) calloc(strlen(msg),sizeof(char));
strcpy(aux,msg);
token=strtok(aux,";");
/* Load the machine */
if(!strcmp(token,"ID")) {
    /*
     * The format of the string is NAME;VM;HOST;CRITICAL;SERVICE1;SERVICE2;...;SERVICEN
     * We suppose that the message is well formatted, as the slave is intended to do so...
     * If not, we have a trouble!
     */
    token=strtok(NULL,";"); //Load whether is a VM or a PM
    if(node_tmp=search_vms_ip(slaveIP)) {
        if(node_tmp->control==-99) {
            node_tmp->control=0;
            send_udp("receivedid",slaveIP);
            /*
             * The VM was broken and now it is on another place
             * We don't want its config to be changed
             */
        }
        else {
            pthread_mutex_lock (&list_mutex);
            delete_vm_ip(slaveIP);
            pthread_mutex_unlock (&list_mutex);
            sprintf(log_string,"Message thread: Deleting VM %s",slaveIP);
            do_log(log_string);
            send_udp("autorestart",slaveIP);
        }
        return;
    }
    else if(pos=search_pos_pms_ip(slaveIP)) {
        /* Delete the machine and load it again */
        pthread_mutex_lock (&list_mutex);
        list_del(pos);
        pthread_mutex_unlock (&list_mutex);
        sprintf(log_string,"Message thread: Deleting machine %s",slaveIP);
        do_log(log_string);
        send_udp("autorestart",slaveIP);
        return;
    }
    node_tmp=(struct node_list *)malloc(sizeof(struct node_list));
    if(strcmp(token,"0"))
        node_tmp->isVM=1; //It's a VM
    else
        node_tmp->isVM=0;
    node_tmp->control=0;
    token=strtok(NULL,";"); //Skip the host field if it's a PM
    if(node_tmp->isVM) {
        node_host_tmp=search_pms_ip(token);
        if(node_host_tmp==NULL) {
            sprintf(log_string,
                "Message thread: Config: Error, I don't know the PM of this VM (%s)!.",slaveIP);
            send_udp("receivedid",slaveIP);
            send_udp("errorvm",slaveIP);
            return;
        }
    }

    node_tmp->host = (char * ) calloc(15 , sizeof(char));
    node_tmp->ip = (char * ) calloc(15 , sizeof(char));
    sprintf(node_tmp->host,"%s",token);
    strcpy(node_tmp->ip,slaveIP); //Load the ip
```

```
token=strtok(NULL,";"); //Get the name
node_tmp->name=(char * ) calloc(strlen(token) , sizeof(char));
strcpy(node_tmp->name,token);
token=strtok(NULL,";"); //Whether is critical or not
node_tmp->critical=no;
if(!strcmp(token,"1"))
    node_tmp->critical=yes;
node_tmp->heartbeat=time(NULL);
node_tmp->count=0;
node_tmp->status=0;

INIT_LIST_HEAD(&node_tmp->services.list);
token=strtok(NULL,";"); // Load the first service
do {
    service_tmp=(struct service_list *)malloc(sizeof(struct service_list));
    service_tmp->service=(char * ) calloc(strlen(token) , sizeof(char));
    strcpy(service_tmp->service,token);
    service_tmp->status=0;
    service_tmp->count=0;
    service_tmp->heartbeat=time(NULL);
    do_log(log_string);
    list_add_tail(&(service_tmp->list), &(node_tmp->services.list));
} while(token=strtok(NULL,";"));
list_for_each(pos, &node_tmp->services.list) {
    service_tmp=list_entry(pos, struct service_list, list);
    strcpy(aux,service_tmp->service);
    token=strtok(aux,"#");
    if(strcmp(token,service_tmp->service)) {
        service_tmp->service = (char *)
                      realloc(service_tmp->service,strlen(token));
        strcpy(service_tmp->service,token);
        #ifdef DEBUG
        if(VERBOSITY>6) {
            while(token=strtok(NULL,"#")) {
                do_log(token);
            }
        }
        #endif
    }
}
free(aux);
if(!node_tmp->isVM) {
    /* It is a PM */
    pthread_mutex_lock (&list_mutex);
    list_add_tail(&(node_tmp->list), &(pms.list));
    node_tmp->vms=(struct node_list *) malloc(sizeof(struct node_list));
    INIT_LIST_HEAD(&(node_tmp->vms->list));
    pthread_mutex_unlock (&list_mutex);
    sprintf(log_string,"Message thread: Loaded config for PM %s",slaveIP);
    do_log(log_string);
}
else {
    /* It is a VM */
    node_tmp->vms=NULL;
    pthread_mutex_lock (&list_mutex);
    list_add_tail(&(node_tmp->list), &(node_host_tmp->vms->list));
    node_host_tmp++;
    pthread_mutex_unlock (&list_mutex);
    sprintf(log_string,"Message thread: Loaded config for VM %s",slaveIP);
    do_log(log_string);
}
pthread_mutex_unlock (&cs_mutex);
```

```
            send_udp("receivedid",slaveIP);
        }
       else if(!strcmp(token,"ok")) {
           node_tmp=search_ip(slaveIP);
           pthread_mutex_lock (&list_mutex);
           node_tmp->status=0;
           pthread_mutex_unlock (&list_mutex);
        }
        else if(!strcmp(token,"SD")) {
            /* We have received a Status Data packet */
            node_tmp=search_ip(slaveIP);
            if(node_tmp==NULL) {
                sprintf(log_string,"Message thread: Error, I don't know this machine (%s).",slaveIP);
                do_log(log_string);
                send_udp("receivedid",slaveIP);
                send_udp("autorestart",slaveIP);
                return;
            }
            token=strtok(NULL,";");
            service_tmp=search_service_ip(token,node_tmp);
            if(service_tmp==NULL) {
                sprintf(log_string,
                    "Message thread: Error, the machine %s does not have the service %s.",slaveIP,token);
                do_log(log_string);
                send_udp("receivedid",slaveIP);
                send_udp("errorsv",slaveIP);
                return;
            }
            pthread_mutex_lock (&list_mutex);
            token=strtok(NULL,";");

            if(atoi(token) == 0 || (service_tmp->status!=-99 && service_tmp->status!=-9)) {
                service_tmp->status=atoi(token);
            }
            service_tmp->heartbeat=time(NULL);
            node_tmp->heartbeat=time(NULL);
            node_tmp->count=0;
            pthread_mutex_unlock (&list_mutex);
            #ifdef DEBUG
            if(VERBOSITY>4 && service_tmp->status!=-9 && service_tmp->status!=-99) {
                sprintf(log_string,
                    "Message thread: Updated service status (%d) for %s",service_tmp->status,service_tmp->serv
                do_log(log_string);
            }
            #endif
        }

    }
    /* Checks if a machine is working */
    int check_machine(struct node_list *node) {
        struct service_list *service_tmp;
        struct node_list *node_tmp, *node_tmp2;
        struct list_head *pos;
        time_t current_time;
        char log_string[MAX_LOG_MSG];
        double diff;
        int control;
        char cmd[500];
        current_time=time(NULL);
        diff=difftime(current_time,node->heartbeat);
        #ifdef DEBUG
        if(VERBOSITY>4) {
```

```
        sprintf(log_string,
            "Check thread: Last heartbeat for NODE %s (%s)was %fs ago",node->name,node->ip,diff);
        do_log(log_string);
    }
    #endif
    if(diff>=DELAY_HEARTBEAT && node->count<MAX_TRY_HEARTBEAT) {
        pthread_mutex_lock (&list_mutex);
        node->count++;
        pthread_mutex_unlock (&list_mutex);
        #ifdef DEBUG
        if(VERBOSITY>4) {
            sprintf(log_string,
                "\t\tand it's too late, incrementing count(%d/%d)",
                node->count,MAX_TRY_HEARTBEAT);
            do_log(log_string);
        }
        #endif
        send_udp("collecteddata",node->ip);
    }
    if(node->count>=MAX_TRY_HEARTBEAT) {
        if(node->isVM) {
            /* The VM machine has failed, so I'll restart it */

            /* Destoy this one an create another one */
            sprintf(cmd,"AC;VM;DESTROY;%s",node->name);
            send_udp(cmd,node->host);
            sprintf(cmd,"AC;VM;CREATE;%s.cfg",node->name);
            send_udp(cmd,node->host),
            pthread_mutex_lock (&list_mutex);
            /* This flag is to tell the master the VM is going to another place */
            node->control=-99;

            pthread_mutex_unlock (&list_mutex);
            sprintf(log_string,
                "Message thread: RESTARTED VM %s on %s",node->ip,node_tmp->ip);
            do_log(log_string);
        }
        else {
            /*
             * The physical machine is broken.
             * At this point we just move the machines to another location
             */
            do_log("Message thread: The Machine %s is BROKEN",node->ip);
            pthread_mutex_lock (&list_mutex);
            node->status=-1;
            list_for_each(pos, &node->vms->list) {
                node_tmp2= list_entry(pos, struct node_list, list);
                /* Search for the new PM */
                node_tmp=search_less_vms(node_tmp2->ip);
                /* Destoy this one an create another one */
                sprintf(cmd,"AC;VM;CREATE;%s.cfg",node_tmp2->name);
                send_udp(cmd,node_tmp->ip),
                pthread_mutex_lock (&list_mutex);
                /* This flag is to tell the master the VM is going to another place */
                node_tmp2->control=-99;
                /* Change the host IP of the VM */
                strcpy(node_tmp2->host,node_tmp->ip);
                /* Delete the machine from the old one */
                delete_vm_ip(node_tmp2->ip);
                /* Add the machine on the new one */
                list_add_tail(&(node_tmp2->list), &(node_tmp->vms->list));
                node_tmp->control++;
```

```
                pthread_mutex_unlock (&list_mutex);
                sprintf(log_string,
                    "Message thread: MOVED VM %s to %s",node_tmp2->ip,node_tmp->ip);
                do_log(log_string);
            }
            pthread_mutex_unlock (&list_mutex);
        }
    return(-1);
    }
    return(0);
}
/* Just checks a service */
int check_service(struct service_list *service,struct node_list *node) {
    char log_string[MAX_LOG_MSG];
    double diff;
    char *cmd;
    #ifdef DEBUG
    if(VERBOSITY>4) {
        sprintf(log_string,
            "Check thread: Status for SERVICE %s is %d",
            service->service,service->status);
        do_log(log_string);
    }
    #endif
    if(service->status==0) {
        pthread_mutex_lock (&list_mutex);
        service->count=0;
        pthread_mutex_unlock (&list_mutex);
    }
    else if(service->status!=-99) {
        if(service->count>=MAX_TRY_HEARTBEAT) {
            cmd=(char *) calloc(14 + strlen(service->service),sizeof(char));
            if(service->status==-9) {
                /* The service is broken. We stop it. */
                sprintf(log_string,
                    "Check thread: SERVICE %s on %s (%s) doesn't work at all. Stop",
                    service->service,node->name,node->ip);
                do_log(log_string);
                sprintf(cmd,"AC;SV;%s;stop",service->service);
                send_udp(cmd,node->ip);
                service->status=-99;
                /* We must notidy the admin */
            }
            else {
                /*
                 * Here we just try to restart the machines
                 */
                sprintf(log_string,
                    "Check thread: SERVICE %s on %s (%s) doesn't work. Restart",
                    service->service,node->name,node->ip);
                do_log(log_string);
                sprintf(cmd,"AC;SV;%s;restart",service->service);
                send_udp(cmd,node->ip);
                service->status=-9;
                service->count=0;
            }
        }
        else {
            pthread_mutex_lock (&list_mutex);
            service->count++;
            pthread_mutex_unlock (&list_mutex);
            #ifdef DEBUG
```

```
            if(VERBOSITY>4) {
                sprintf(log_string,
                    "\t\tand it doesn't work, incrementing count(%d/%d)",
                    service->count,MAX_TRY_HEARTBEAT);
                do_log(log_string);
            }
            #endif
        }
    }
}
/* This function periodically checks the status of the clients */
void* check_function (void* arg) {
    struct node_list *node_tmp, *vms_tmp;
    struct service_list *service_tmp;
    struct list_head *node_pos, *vms_pos, *service_pos, *tmp_pos;
    time_t current_time;
    pthread_mutex_lock (&cs_mutex);
    while(1) {
        /* Check each node */
        list_for_each(node_pos, &pms.list){
            node_tmp= list_entry(node_pos, struct node_list, list);
            if(!check_machine(node_tmp)) {
                /* If the machine works, we check the VMS */
                if(!node_tmp->isVM) {
                    vms_tmp=node_tmp->vms;
                     list_for_each_safe(vms_pos,tmp_pos, &node_tmp->vms->list){
                         vms_tmp=list_entry(vms_pos, struct node_list, list);
                         if(vms_tmp->control!=-99)
                             check_machine(vms_tmp)
                     }
                }
                /* And check the services */
                list_for_each(service_pos,&node_tmp->services.list) {
                    service_tmp=list_entry(service_pos,struct service_list,list);
                    check_service(service_tmp,node_tmp);
                }
            }
        }
        sleep(1);
    }
    return(NULL);
}
/* This the message processing thread */
void* message_function (void* arg) {
    int i,sd, rc, n, cliLen;
    struct sockaddr_in cliAddr, servAddr;
    char msg[MAX_MSG];
    char log_string[MAX_LOG_MSG];
    char slaveIP[15];
    /* Socket creation */
    sd=socket(AF_INET, SOCK_DGRAM, 0);
    if(sd<0) {
        do_log("Message thread error: cannot open socket.");
        exit(1);
    }
    /* Local ports linking */
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(MASTER_SERVER_PORT);
    rc = bind (sd, (struct sockaddr *) &servAddr,sizeof(servAddr));
    if(rc<0) {
        sprintf(log_string,"Message thread: cannot bind port number %d.", MASTER_SERVER_PORT);
```

```
            do_log(log_string);
            exit(1);
        }
        sprintf(log_string,"Message thread: waiting for data on port UDP %u.",MASTER_SERVER_PORT);
        do_log(log_string);
        while(1) {
            /*  buffer init */
            memset(msg,0x0,MAX_MSG);
            /* Start receiving messages */
            cliLen = sizeof(cliAddr);
            n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) &cliAddr, &cliLen);
            if(n<0) {
                sprintf(log_string,"Message thread: cannot receive data.");
                do_log(log_string);
                continue;
            }

            /*Message managment */
            #ifdef DEBUG
            if (VERBOSITY>4) {
                sprintf(log_string,
                       "Message thread: received message from %s port %u (%s).",
                       inet_ntoa(cliAddr.sin_addr),ntohs(cliAddr.sin_port),msg);
                do_log(log_string);
            }
            #endif
            strncpy(slaveIP,inet_ntoa(cliAddr.sin_addr),15);
            parse(msg,slaveIP);
        }
    return NULL;
    }
    int main(int args, char* argg[]) {
        int i;
        int a;
        char log_string[MAX_LOG_MSG];
        pthread_t message_thread;
        pthread_t check_thread;
        pthread_t poll_thread;
        /* Initilise the list */
        INIT_LIST_HEAD(&pms.list);
        a=pthread_create(&message_thread,NULL,&message_function,NULL);
        if (a==0) {
            do_log("Message thread: Thread started.");
        }
        else {
            do_log("Message thread: Thread cannot start.");
        }

        /* This is the status checking thread */
        pthread_mutex_lock (&cs_mutex);
        a=pthread_create(&check_thread,NULL,&check_function,NULL);
        if (a==0) {
            do_log("Check thread: Thread started.");
        }
        else {
            do_log("Check thread: Thread cannot start.");
        }
        pthread_join(message_thread,NULL);
        return 0;
    }
```

### C.2.2 slave.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <malloc.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include "control.h"
#include "list.h"
struct machine_data {
    char master[15];
    char *name;
    char *scripts_location;
    enum {pm=0,vm=1} type;
    char host[15];
    char *services;
    enum {no=0,yes=1} critical;
    int delay;
} machine_data;

struct service_list {
    char * script;
    char * depends_on;
    int status;
    struct list_head list;
};
struct service_list services;
/* exit type flag */
int exitflag;
int handshaking;
/* mutex a protezione del logging su file*/
pthread_mutex_t log_mutex = PTHREAD_MUTEX_INITIALIZER;
/* mutex a protezione dell'uscita degli UDP */
pthread_mutex_t udp_out_mutex =  PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t handshaking_mutex =  PTHREAD_MUTEX_INITIALIZER;
struct service_list * search_service(char* service) {
    struct list_head *pos;
    struct service_list *tmp;
    list_for_each(pos,&services.list) {
        tmp=list_entry(pos,struct service_list,list);
        if(!strcmp(tmp->script,service))
            return(tmp);
    }
    return(NULL);
}
/* Function to log the messages */
void do_log(char * message) {
    FILE * logfile;
    time_t current_time;
    char *line;
    line =(char *) calloc (30+strlen(message),sizeof(char));
    current_time=time(NULL);
    strncpy(line,ctime(&current_time),24);
    line[24]='\0';
```

```
    strcat(line," ");
    strcat(line,message);
    pthread_mutex_lock (&log_mutex);
    if(strstr(line,"\n")==NULL)
        fprintf(stderr,"HC: %s\n",line);
    else
        fprintf(stderr,"HC: %s",line);
    pthread_mutex_unlock (&log_mutex);
}
/* Function to remove whitespaces from the options */
void trim(char *buf) {
    char *p1;
    char *p2;
    int lc=strlen(buf);
    p1=buf+lc;

    p1--;
    while ((p1[0]<=0x20) && (p1>buf)) {
        p1[0]=0x0;
        p1--;
    }
    p1=buf;
    p2=buf;

    while (p2[0]) {
        if (p2[0]>0x20) { break; }
        p2++;
    }

    while (p2[0]) {
        p1[0]=p2[0];
        p1++; p2++;
    }
    p1[0]=0x0;
}
/* Check and load the config file */
int check_config(void) {
    FILE *fp;
    char * line;
    char log_string[MAX_LOG_MSG];
    char *option,*value,*aux, *aux2, *aux3;
    struct service_list *tmp;
    line=(char *) calloc(MAX_CONFIG_LINE,sizeof(char));
    option=(char *) calloc(MAX_CONFIG_LINE,sizeof(char));
    value=(char *) calloc(MAX_CONFIG_LINE,sizeof(char));
    /* Some values */
    machine_data.scripts_location="./hc.scripts/";
    sprintf(machine_data.host,"0.0.0.0");
    machine_data.type=pm;
    machine_data.delay=5;
    machine_data.critical=no;
    if((fp = fopen(CONFIG_FILE, "r"))==NULL){
        sprintf(log_string,"Config: Error opening file %s.",CONFIG_FILE);
        do_log(log_string);
        return(1);
    }
    while(!feof(fp) && fgets(line,MAX_CONFIG_LINE,fp)!=NULL) {
        trim(line);
        if((line[0]!='#') && (line[1]!='\0')) {
            #ifdef DEBUG
            if(VERBOSITY>1) {
                sprintf(log_string,"Config: Got option: %s",line);
```

```
    do_log(log_string);
}
#endif
option=strtok(line,"=");
value=strtok(NULL,"=");
trim(option);
trim(value);

/* Loading the options */
if(value==NULL) {
    sprintf(log_string,"Config: Empty option %s.",option);
    do_log(log_string);
    return(1);
}
else if(!strcmp(option,"name")) {
    machine_data.name=(char *) calloc(strlen(value),sizeof(char));
    strcpy(machine_data.name,value);
}
else if(!strcmp(option,"scripts-location")) {
    machine_data.scripts_location=(char *) calloc (strlen(value),sizeof(char));
    strcpy(machine_data.scripts_location,value);
}
else if(!strcmp(option,"type")) {
    if(!strcmp(value,"pm"))
        machine_data.type=pm;
    else
        machine_data.type=vm;
}
else if(!strcmp(option,"master")) {
    strcpy(machine_data.master,value);
}
else if(!strcmp(option,"delay")){
    machine_data.delay=atoi(value);
}
else if(!strcmp(option,"service")) {
    machine_data.services=(char *) calloc( strlen(value),sizeof(char));
    strcpy(machine_data.services,value);
}
else if(!strcmp(option,"critical")) {
    if(strcmp(value,"yes"))
        machine_data.critical=yes;
    else
        machine_data.critical=no;
}
else if(!strcmp(option,"host")) {
    /* If we have a host option, the machine must be a virtual one */
    if(machine_data.type) {
        strcpy(machine_data.host,value);
    }
    else {
        sprintf(log_string,
            "Config: Incompatible option type=pm; host=%s.",
             option);
        do_log(log_string);
        return(1);
    }
}
else {
    sprintf(log_string,"Config: Unknown option %s.",option);
    do_log(log_string);
    return(1);
}
```

```
        #ifdef DEBUG
        if(VERBOSITY>4) {
            sprintf(log_string,
                "Config: Loaded option: machine_data.%s=%s",
                option,value);
            do_log(log_string);
        }
        #endif
    }
}
if(fclose(fp)!=0) {
     sprintf(log_string,"Config: Error closing file %s.",CONFIG_FILE);
     do_log(log_string);
     return(1);
}
/* Check if everything is ok */
if(machine_data.type && !strcmp(machine_data.host,"0.0.0.0")) {
    do_log("Config: Error, type=vm needs a host option!");
    return(1);
}
if(machine_data.services==NULL) {
    do_log("Config: Error, you must provide at least one service to check");
    return(1);
}
else {
    /*
     * Ok, everything is right. Now load the values on the list.
     *
     * Please note that we check if the scripts exist here becasuse the options
     * would be in the wrong order in the config file.
     *
     */
    INIT_LIST_HEAD(&services.list);
    aux=(char *) calloc(strlen(machine_data.services),sizeof(char));
    strcpy(aux,machine_data.services);
    value=strtok(aux,";");
    do {
        aux3=(char *) calloc(strlen(value),sizeof(char));
        /* Take out the services it depends on */
        int i = strcspn(value,"#");
        strncpy(aux3,value,i);
        tmp=(struct service_list *) malloc(sizeof(struct service_list));
        tmp->script=(char * ) calloc(strlen(aux3) , sizeof(char));
        strcpy(tmp->script,aux3);
        tmp->status=0;
        free(aux3);
        /* Check */
        aux2=(char *) calloc(strlen(tmp->script) +
             strlen(machine_data.scripts_location),sizeof(char));
        strcat(aux2,machine_data.scripts_location);
        strcat(aux2,tmp->script);
        if(!access(aux2,X_OK)) {
            list_add_tail(&(tmp->list), &(services.list));
        }
        else {
            sprintf(log_string,
                "Config: Error, the script %s either not exist or is not executable",
                aux2);
            do_log(log_string);
            return(1);
        }
    } while(value=strtok(NULL,";"));
```

```
    }
    return(0);
}
/* Function to send a UDP message to the master */
void send_udp(char * message) {
    int sd, rc;
    struct sockaddr_in cliAddr, remoteServAddr;
    struct hostent *h;
    #ifdef DEBUG
    char *log_string;
    #endif
    pthread_mutex_lock (&udp_out_mutex);
    h = gethostbyname(machine_data.master);
    if(h==NULL) {
        do_log("Udp gateway: Server unknown.");
        return;
    }
    remoteServAddr.sin_family = h->h_addrtype;
    memcpy((char *) &remoteServAddr.sin_addr.s_addr, h->h_addr_list[0], h->h_length);
    remoteServAddr.sin_port = htons(MASTER_SERVER_PORT);
    sd = socket(AF_INET,SOCK_DGRAM,0);
    if(sd<0) {
        do_log("Udp gateway: Cannot open socket.");
        return;
    }

    cliAddr.sin_family = AF_INET;
    cliAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    cliAddr.sin_port = htons(0);

    rc = bind(sd, (struct sockaddr *) &cliAddr, sizeof(cliAddr));
    if(rc<0) {
        do_log("Udp gateway: Cannot bind port.");
        return;
    }
    rc = sendto(sd, message, strlen(message)+1, 0,
           (struct sockaddr *) &remoteServAddr, sizeof(remoteServAddr));
    if(rc<0) {
        do_log("Udp gateway: Cannot send data.");
        close(sd);
        return;
    }
    #ifdef DEBUG
    else {
        if (VERBOSITY>4) {
            log_string = (char *) calloc (40+strlen(message),sizeof(char));
            sprintf(log_string,"Udp gateway: Sending message (%s)",message);
            do_log(log_string);
        }
    }
    #endif
    close (sd);
    pthread_mutex_unlock (&udp_out_mutex);
    return;
}
/* Just send the ID */
void send_id(void) {
    char *aux =(char *) calloc(3+
                strlen(machine_data.name)+
                strlen(machine_data.services)+
                15+ // IP
                15+ // Host
```

```
            1+  // PM/VM
            1,  // Critical
            sizeof(char));
    sprintf(aux,"ID;%o;%s;%s;%o;%s",machine_data.type,
                   machine_data.host,
                   machine_data.name,
                   machine_data.critical,
                   machine_data.services);
    send_udp(aux);
}
/* funzione che esegue il parsing dei messaggi ricevuti via UDP */
int parse(char * msg) {
    int i,j;
    char log_string[MAX_LOG_MSG];
    struct service_list *tmp;
    struct list_head *pos;
    char * aux, * token, * id, * cmd;
    /*
     * Valori di ritorno:
     * -1   -> Fare un restart
     * 1    -> Non fare nulla
     * -99  -> Comando non conosciuto
     */
    /* Is anything wrong? */
    if (!strcmp(msg,"autorestart")) {
        send_udp("autorestart-start");
        return(-1);
    }
    /* First, we wait for the handshaking */
    else if(!handshaking) {
        if(!strcmp(msg,"receivedid")) {
            handshaking=1;
            do_log("Message thread: Master received ID. We're ready");
            return(1);
        }
    }
    /* I have to send the data */
    else if (!strcmp(msg,"collecteddata")) {
        list_for_each(pos, &services.list){
            tmp= list_entry(pos, struct service_list, list);
            aux = (char *) calloc(strlen(tmp->script) +4,sizeof(char)+sizeof(int));
            sprintf(aux,"SD;%s;%d",tmp->script,tmp->status);
            send_udp(aux);
            free(aux);
        }
        return(0);
    }
    /* I'm ok */
    else if (!strcmp(msg,"status")) {
        send_udp("ok");
        return(0);
    }
    /* In case the master needs me to send my id again */
    else if(!strcmp(msg,"sendid")){
        send_id();
        return(1);
    }
    else if(!strcmp(msg,"errorvm")) {
        do_log("Message thread: Misconfiguration error... Exiting.");
        return(-1);
    }
    /* Aqui van las acciones */
```

```
        else {
            token=(char *) calloc(strlen(msg),sizeof(char));
            id=(char *) calloc(strlen(msg),sizeof(char));
            aux=(char *) calloc(strlen(msg),sizeof(char));
            cmd=(char *) calloc(strlen(msg),sizeof(char));
            strcpy(aux,msg);
            token=strtok(aux,";");
            if(!strcmp(token,"AC")) {
                token=strtok(NULL,";");
                int value;
                if(!strcmp(token,"REBOOT")) {
                    do_log("DEBUG: REINICIO");
//                  value=system("init 6");
//                  sprintf(aux,"%s;%d",msg,value);
//                  send_udp(aux);
                }
                else if(!strcmp(token,"VM")) {
                    if(machine_data.type==pm) {
                        token=strtok(NULL,";");
                        id=strtok(NULL,";");
                        if(!strcmp(token,"SHUTDOWN")) {
                            sprintf(cmd,"xm shutdown %s",id);
                        }
                        else if(!strcmp(token,"DESTROY")) {
                            sprintf(cmd,"xm destroy %s",id);
                        }
                        else if(!strcmp(token,"REBOOT")) {
                            sprintf(cmd,"xm reboot %s",id);
                        }
                        else if(!strcmp(token,"CREATE")) {
                            sprintf(cmd,"xm create %s",id);
                        }
                        else if(!strcmp(token,"MIGRATE")){
                            sprintf(cmd,"xm migrate %s",id);
                        }
                        else if(!strcmp(token,"LMIGRATE")){
                            sprintf(cmd,"xm migrate -l %s",id);
                        }
                        value=system(cmd);
                        sprintf(aux,"%s;%d",msg,value);
                        send_udp(aux);
                    }
                    else {
                        do_log("Message thread: Received to do an action over a VM and I'm a VM!");
                        send_udp("AC;VM;NOT");
                        return(-99);
                    }
                }
                else if(!strcmp(token,"SV")){
                    token=strtok(NULL,";");
                    id=strtok(NULL,";");
                    tmp=search_service(token);
                    aux = (char *) calloc(strlen(tmp->script) +
                          strlen(machine_data.scripts_location) + 8,sizeof(char));
                    strcat(aux,machine_data.scripts_location);
                    strcat(aux,tmp->script);
                    strcat(aux," ");
                    strcat(aux,id);
                    sprintf(log_string,"Check Status thread: Executing action %s", aux);
                    do_log(log_string);
                    system(aux);
                    if(value==0)
```

```
                    sprintf(log_string,"Check Status thread: %s OK",aux);
                else
                    sprintf(log_string,"Check Status thread: %s ERR",aux);
                do_log(log_string);
                sprintf(aux,"%s;%d",msg,value);
                send_udp(aux);
                do_log(aux);
                free(aux);
            }
            else {
                #ifdef DEBUG
                if (VERBOSITY>2) {
                    sprintf(log_string,"Message thread: unknown command %s.", msg);
                    do_log(log_string);
                }
                #endif
                return(-99);
            }
        }
        else {
            #ifdef DEBUG
            if (VERBOSITY>2) {
                sprintf(log_string,"Message thread: unknown command %s.", msg);
                do_log(log_string);
            }
            #endif
            return(-99);
        }
    }
}
/* Function to process the UDP messages */
void* message_function (void* arg) {
    int control;
    int sd, rc, n, cliLen;
    struct sockaddr_in cliAddr, servAddr;
    char msg[MAX_MSG];
    char log_string[MAX_LOG_MSG];
    /* Socket creation */
    sd=socket(AF_INET, SOCK_DGRAM, 0);
    if(sd<0) {
        do_log("Message thread error: cannot open socket.");
        exit(1);
    }
    /* Local port connection */
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(SLAVE_PORT);
    rc = bind(sd, (struct sockaddr *) &servAddr,sizeof(servAddr));
    if(rc<0) {
        sprintf(log_string,"Message thread: cannot bind port number %d.", SLAVE_PORT);
        do_log(log_string);
        exit(1);
    }
    sprintf(log_string,"Message thread: waiting for data on port UDP %u.",SLAVE_PORT);
    do_log(log_string);
    control=99;
    while(1) {
        if (control==-1) {
            close(sd);
            sleep(5);
            pthread_exit((void *)control);
        }
```

```
        else if(control==-42) {
            close(sd);
            sleep(5);
            pthread_exit((void *)control);
        }

        /* Buffer inicialization */
        memset(msg,0x0,MAX_MSG);
        /* Message reception */
        cliLen = sizeof(cliAddr);
        n = recvfrom(sd, msg, MAX_MSG, 0, (struct sockaddr *) &cliAddr, &cliLen);
        if(n<0) {
            sprintf(log_string,"Message thread: cannot receive data.");
            do_log(log_string);
            continue;
        }

        #ifdef DEBUG
        if (VERBOSITY>4) {
            sprintf(log_string,"Message thread: received message from %s port %u (%s).",
                inet_ntoa(cliAddr.sin_addr),ntohs(cliAddr.sin_port),msg);
            do_log(log_string);
        }
        #endif
        control=parse(msg);
    }
    return NULL;
}
/* Executes the scripts, collects the result, and send an UDP to the master */
void* checkstatus_function (void* arg) {
    char log_string[MAX_LOG_MSG];
    struct service_list *tmp;
    struct list_head *pos;
    char * aux;
    int delay;
    int value;
    /* We can't send anything if the master doesn't know us! */
    pthread_mutex_lock(&handshaking_mutex);
    while(1) {
        /* We traverse the list and execute each script. */
        list_for_each(pos, &services.list){
            tmp= list_entry(pos, struct service_list, list);
            aux = (char *) calloc(strlen(tmp->script) +
                  strlen(machine_data.scripts_location),sizeof(char));
            strcat(aux,machine_data.scripts_location);
            strcat(aux,tmp->script);
            sprintf(log_string,"Check Status thread: Executing script %s", aux);
            do_log(log_string);
            value=system(aux);
            if(value==0)
                sprintf(log_string,"Check Status thread: %s OK",aux);
            else
                sprintf(log_string,"Check Status thread: %s ERR",aux);
            do_log(log_string);
            tmp->status=value;
            sprintf(aux,"SD;%s;%d",tmp->script,tmp->status);
            send_udp(aux);
            free(aux);
        }
        sleep(machine_data.delay);
    }
    return NULL;
```

```
}
void* action_function (void* arg) {
    char log_string[MAX_LOG_MSG];
    int delay;
    handshaking=0;
    /* Make sure the master know us */
    while(!handshaking) {
        send_id();
        sleep(5);
    }
    /* Unlock the mutex, so everything can start working */
    pthread_mutex_unlock(&handshaking_mutex);
    return NULL;
}
int main(int args, char* argv[]) {
    int i;
    int a;
    int exitflag;
    char log_string[MAX_LOG_MSG];
    pthread_t message_thread;
    pthread_t checkstatus_thread;
    pthread_t action_thread;
    pthread_attr_t checkstatus_attr;
    pthread_attr_t action_attr;
    if(check_config()) {
        return(1);
    }
    a=pthread_create(&message_thread,NULL,&message_function,NULL);
    if (!a) {
        do_log("Message thread: Thread started.");
    }
    else {
        do_log("Message thread: Thread cannot start.");
    }
    /* We need to lock the Checkstatus thread until Action Thread unlocks the MUTEX */
    pthread_mutex_lock(&handshaking_mutex);
    pthread_attr_init (&checkstatus_attr);
    pthread_attr_setdetachstate(&checkstatus_attr, PTHREAD_CREATE_DETACHED);
    a=pthread_create(&checkstatus_thread,&checkstatus_attr,&checkstatus_function,NULL);
    if (!a) {
        do_log("Check Status thread: Thread started.");
    }
    else {
        do_log("Check Status thread: Thread cannot start.");
    }
    pthread_attr_init (&action_attr);
    pthread_attr_setdetachstate(&action_attr, PTHREAD_CREATE_DETACHED);
    a=pthread_create(&action_thread,&action_attr,&action_function,NULL);
    if (!a) {
        do_log("Action thread: Thread started.");
    }
    else {
        do_log("Action thread: Thread cannot start.");
    }

    pthread_join(message_thread, (void **)&exitflag);
    if (exitflag==-1) {
        do_log("Main program: Restarting ... ... ...");
        do_log("... ...");
        do_log("... ...");
        do_log("... ...");
        execve("./slave",NULL,NULL);
```

```
    }

    return 0;
}
```

### C.2.3 control.h

```
#define SLAVE_PORT 3001
#define MASTER_SERVER_PORT 3000
#define MAX_MSG 65536
#define MAX_LOG_MSG 1024
#define MAX_CONFIG_LINE 1024
#define MAX_THREADS 5
#define NUMCLIENTS 2
#define VERBOSITY 5 /* 1-5 */
#define DEBUG
#define CONFIG_FILE "hc.slave.config"
#define DELAY_HEARTBEAT 5
#define MAX_TRY_HEARTBEAT 10
```

### C.2.4 File di congurazione

```
# Config File for hc-slave
#################
# Basic options #
#################
# IP address of the master
master=192.168.1.1
# Hostname of the machine (or name of the vm)
name=prova
# Delay (in seconds) between checks are performed
delay=10
# The path to the scripts (slash ended)
scripts-location=./hc-scripts/
# Put 'pm' if this is a physical machine or 'vm' for Xen virtual machines
# If you don't know what the hell is Xen or a Virtual Machine, leave it on 'pm'
type=pm
# Join services by a comma ','. A consequent script under 'scripts-location' must exist
service=mail;apache
# Either this machine is critical or not
#critical=no
####################
# Advanced options #
####################
# If type is set to 'vm' this must contain the host's IP address
#host=127.0.0.1
```

# Bibliografia


[1] Foster, Ian. What is the Grid? A Three Point Checklist. Argonne National Laboratory & University of Chicago (2002).
URL: `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf`

[2] CMS Computing Project. CMS, The Computing Project. Technical Design Support. CERN/LHCC 2005-013 (Giugno 2005).

[3] Knobloch, J., Robertson, L. et al. LHC Computing Project. Technical Design Report CERN/LHCC 2005-024 (Giugno 2005).
URL: `http://cern.ch/lch`

[4] Neri, Igor. Tesi di Laurea: “Installazione, configurazione e monitoraggio di un sito INFN-Grid”. Università degli studi di Perugia (2005).
URL: `http://cms.pg.infn.it/`

[5] Popek, G.J., Goldberg, R.P. Formal requirements for virtualizable third generation architectures. Communications of the ACM, Volume 17 - Issue 7 (Luglio 1974).
URL: `http://portal.acm.org/citation.cfm?doid=361011.361073`

[6] Smith, J., Ravi, N. Virtual Machines. Morgan Kaufmann (2005).

[7] Xen homepage.
URL: `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/index.html`

[8] SAN
URL: `http://en.wikipedia.org/wiki/Storage_area_network`

[9] SCSI
URL: `http://en.wikipedia.org/wiki/SCSI`

[10] Fibre Channel Overview
URL: `http://hsi.web.cern.ch/HSI/fcs/spec/overview.htm`

[11] GNBD Homepage
URL: `http://sources.redhat.com/cluster/gnbd/`

[12] IBM GPFS
URL: `http://www-03.ibm.com/servers/eserver/clusters/software/gpfs.html`

[13] Red Hat GFS
URL: `http://www.redhat.com/software/rha/gfs/`

[14] Satran, J., Meth, K. et al. RFC 3720: Internet Small Computer Systems Interface (iSCSI). IETF (Aprile 2004).
URL: `http://www.ietf.org/rfc/rfc3720.txt`

[15] Rockwood, Ben. A Quick Guide to iSCSI on Linux. (Agosto 2004).
URL: `http://cuddletech.com/articles/iscsi/iscsiref.pdf`

[16] core-iSCSI homepage.
URL: `ttp://www.kernel.org/pub/linux/utils/storage/iscsi/`

[17] core-iSCSI HOWTO.
URL: `http://www.kernel.org/pub/linux/utils/storage/iscsi/HOWTO`

[18] The iSCSI Enterprise Target homepage.
URL: `http://iscsitarget.sourceforge.`

[19] Nagios homepage
URL: `http://www.nagios.org/`

[20] Linux-HA homepage
URL: `http://www.linux-ha.org/`

# Glossario

**Berkeley Database II (BDII)**

Database che memorizza lo stato delle risorse è che viene interrogato nel momento incui un RB fa una richiesta.

**Computing Element (CE)**

Elemento verso il quale si accede a una farm sottomettendo i job ai WN.

**CERN**

Il CERN, sito a Ginevra, è il acronimo di "Organizzazione Europea per la Ricerca Nucleare" (originalmente "Conseil Européen pour la Recherche Nucléaire"), cioè il centro di ricerca di Fisica di particele più importante del mondo

**dom-0**

Dominio Zero. Prima macchina virtuale, con privilegi speziale, caricata da Xen

**dom-U**

Dominio Unprivileged. Macchina virtuale senza privilegi

**Fibre Channel (FC)**

Dispositivo hardware per essere usato in un SAN.

**Global File System (GFS)**

Filesystem distribuito della Red Hat.

**Global Network Block Device (GNBD)**

Protocollo per esportare block device tra una rete.

**General Parallel File System (GPFS)**

Filesystem distribuito della IBM.

**General Public License (GPL)**

Licenza libera della Free Software Foundation.

**Grid**

Paradigma di computazione distribuita nel quale tutti le risorse di un numero indeterminato di computer geograficamente distribuiti vengono inglobati per essere trattati come un unico supercomputer in manera trasparente per gli utenti

**guest**

Un sistema guest, nel'ambito della virtualizzazione, è un sistema virtualizzato.

**host**

Un sistema host, nel'ambito della virtualizzazione, è il sistema che contiene distinte macchine virtuali.

**hypervisor**

(anche Virtual Machine Monitor) è il programma incaricato di eseguire distinte macchine virtuali in un sistema.

**Internet Engineering Task Force (IETF)**

Comunità aperta di tecnici specialisti e ricercatori interessati all'evoluzione tecnica e tecnologica di Internet.

**Istituto Nazionale di Fisica Nucleare (INFN)**

L'Infn è l'ente dedicato allo studio dei costituenti fondamentali della materia e svolge attività di ricerca teorica e sperimentale nei campi della fisica subnucleare nucleare e astroparticellare.

**INFN-Grid**

Struttura di Grid Computing del INFN.

**initiator**

Nel protocolo SCSI (quindi anche in iSCSI) un initiator e qualsiasi dispositivo che fa una richiesta a un target.

**Internet SCSI (iSCSI)**

Protocollo basato in iSCSI per esportare block device tra una rete.

**iSCSI Qualified Name**

Nome univoco per identificare gli elementi di un sistema iSCSI

**World LHC Computing Grid (LCG)**

Intorno di computazione distribuita per il proceso di dati procedenti di esperimenti della Fisica.

**Large Hadron Collider (LHC)**

Nuovo acceleratore di particelle in costruzione presso il CERN di Ginevra per collisioni tra protoni e tra ioni pesanti.

**macchina virtuale**

Macchina che gira su un sistema de virtualizzazione, isolata della macchina fisica

**Resource Broker (RB)**

Elemento incaricato della sottomissione di job verso un determinato CE conservando lo stato attuale del job.

**Request For Comments (RFC)**

Documento che riporta informazioni o specifiche riguardanti nuove ricerce innovazioni e metodologie dell'ambito d'internet.

**Storage Area Network (SAN)**

Sispositivi di memorizzazione condivisi tra una rete di altà velocità.

**Scientific Linux**

Distribuzione Linux –sviluppata dal Fermilab e dal CERN– orientata al ambito scientifico

**Small Systems Computer Interface (SCSI)**

Interfaccia standard progettata per realizzare il trasferimento di dati fra diversi dispositivi.

**Storage Element (SE)**

Componente che si occupa della memorizzazione l'acceso e la replica delle informazioni.

**target**

Nel protocolo SCSI (quindi anche in iSCSI) un target e qualsiasi dispositivo che è oggeto di una richiesta fatta da un initiator.

**User Interface (UI)**

Macchina che serve da interfaccia –per sottomettere i job– fra la Grid e l'utente.

**Macchine Virtuali (VM)**

Vedere "macchina Virtuale".

**Virtual Machine Monitor (VMM)**

Vedere “hypervisor”

**Worker Node (WN)**

Macchine che fanno i calcoli necessari per un determinato job.

**Xen**

Xen è un hypervisor basato nella paravirtualizzazione.