UNIVERSITÀ DEGLI STUDI DI PARMA
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea triennale

Nodi di calcolo virtuali on-demand in ambiente Grid

Candidato:
Marco Grossi

Relatore:
Prof. Roberto Alfieri

Correlatore:
Davide Salomoni

Anno Accademico 2006/2007

Ai miei genitori Norma e Giorgio,
ai miei fratelli Paolo e Diego.

Indice

Capitolo 1

Introduzione

L'avvento del Grid Computing modificherà il modo di pensare alla potenza di calcolo: non è più quantificata nel numero di calcolatori che un centro di ricerca possiede, ma in quanti calcolatori il centro può raggiungere ed utilizzare; tali calcolatori sono sparsi in ogni parte del Mondo, connessi da una rete, magari Internet.
Non solo i centri di ricerca, ma chiunque potrà utilizzare tali risorse, e non solo queste: storage, webservice, ...; con un unico certificato di sicurezza personale si potra accedere a tutto ciò.
Ogni utente ha le proprie esigenze, sia software che hardware; il Grid Computing introduce l'astrazione delle risorse: un utente deve avere solo ciò che vuole, i soliti strumenti con i quali è abituato a lavorare; non deve vedere nient'altro, tantomeno gli altri utenti che richiedono le stesse risorse. Dunque l'astrazione deve arrivare fino ai sistemi operativi ed ai nodi di calcolo; in una parola: virtualizzazione.
I worker node virtuali on-demand sono una risposta.
Il testo è organizzato in tre parti principali:

I Contestualizzazione: si introducono i concetti di Grid Computing e virtualizzazione; si indaga inoltre un caso reale di centro di calcolo di grandi dimensioni;

II Studio di fattibilità: si indagano le modalità di virtualizzazione di un nodo di calcolo ed i problemi connessi alle scelte effettuate; inoltre si dà ampio spazio alle ottimizzazioni relative alla componente storage, sia nel caso di centri di calcolo di grandi dimensioni che piccole;

III Sviluppo: si forniscono i dettagli per la realizzazione di un gestore di nodi di calcolo virtuali on-demand e l'esecuzione di test di scalabilità.

1.1 Ringraziamenti

Alla mia famiglia, per esteso Diego, Giorgio, Kira, Norma e Paolo, che mi ha lasciato libero di intestardirmi nelle mie convinzioni.
A tutti i miei parenti.
La mia avventura universitaria è iniziata a Cremona, assieme agli Infobassa user, ovvero i grandissimi Cippi, Eleonora, Lavez, Lisa, Marcy, Medio, Totti e Zilli.
Non è stato facile cambiare, soprattutto perchè in famiglia avranno un semplice dottore al posto di un altisonante ingegnere, ma ora come ora non posso che esserne convinto della scelta; in quel di Parma ringrazio con ammirazione Alberto, Alessandro, Chietto, Elena, Fabrizio, Luca, Nando, Paolino, Stefano e Vigno.
Un ringraziamento ruggente a Silvia ed a "tu sai chi".
Agli amici, che ultimamente ho decisamente trascurato, Adi, Caccia, Ghiro, Medio, Nene, Panda, Pavel, Ponga, Ross, Vale e Vespa.
A tutti i Prof., Roberto Alfieri, Roberto Bagnara, Egidio Battistini, Federico Bergenti, Marco Bramanti, Roberto Covati, Francesco Di Renzo, Mauro Diligenti, Giulia Ghidini, Grazia Lotti, Carlo Domenico Pagani, Andrea Pescetti, Gabriele Ricci, Gianfranco Rossi ed Enea Zaffanella.
All'LCA lab, Roberto Alfieri, Alessio Cavalieri e Roberto Covati.
Al CNAF, Andrea Chierici, Luca Dell'Agnello, Massimo Donatelli, Guido Guizzunti, Alessandro Italiano, Barbara Martelli, Davide Salomoni e Vladimir Sapunenko.
All'INFN Genova, Alessandro Brunengo e Mirko Corosu.
A tutti coloro che hanno cercato di tirare fuori il meglio di me.
A tutti coloro che mi hanno sopportato, e che dovranno sorbirmi,

Grazie.

Parte I

Contestualizzazione

Capitolo 2

Grid computing

2.1 Il concetto

Accesso ubiquitario a risorse di calcolo e storage: questa una delle possibili definizioni del concetto elaborato da Ian Foster e Carl Kesselman in [1].
"Grid" come griglia di distribuzione dell'energia elettrica, ovvero una rete pervasiva con profonde analogie con la Rete delle reti: Internet.
Risorse di calcolo collegate attraverso una rete con l'uso di protocolli standardizzati ed interfacce compatibili costituiscono una griglia computazionale; questa è una delle applicazioni del grid computing, la quale prevede l'esecuzione di job di calcolo con il ritorno del risultato o un errore.
Per visualizzare meglio questi concetti si pensi alle risorse di calcolo presenti presso enti di ricerca, università, consorzi ed aziende, sia in Italia che all'Estero, sia pubblici che privati: senza una griglia, ognuno fa da se ed utilizza le sue risorse interne. Può capitare che il sistema sia completamente scarico per la maggior parte della sua vita operativa, oppure che non riesca a soddisfare i picchi di lavoro, con conseguente aumento dei tempi di attesa. Per non parlare dei costi di acquisto e manutenzione, a volte proibitivi se non in presenza di finanziamenti pubblici. L'introduzione di una computational grid porterebbe i seguenti vantaggi ai soggetti interessati:

- un soggetto con fondi limitati oppure necessità stagionali potrebbe decidere di non allestire un proprio centro di calcolo, ma utilizzare le risorse in outsourcing, pagando in base all'utilizzo effettivo;
- un soggetto proprietario di un centro di calcolo può colmare l'inattività e le spese di manutenzione fornendo accesso alle sue risorse a pagamento, oltre a limitare gli inconvenienti nel caso di picchi di lavoro accedendo alle risorse di altri soggetti.

L' "accedere alle risorse di altri soggetti" può prevedere, ad esempio, che un centro di ricerca italiano possa utilizzare le risorse rese disponibili da un ente giapponese, poiché i Paesi interessati hanno stretto un accordo di mutuo utilizzo delle risorse di calcolo.

2.2 Utenti e Virtual Organization

Attualmente non esiste una computational grid di "default", ovvero un mercato nella quale si incontrano domanda e offerta; si firmano partnership fra i soggetti interessati.
Etichettiamo questi soggetti con il termine Virtual Organization, nel seguito per brevità V.O..
Una V.O. può essere un soggetto:

- attivo(concede in uso le proprie risorse),
- passivo(utilizza le risorse di V.O. attive) oppure
- entrambe.

Partnership fra varie V.O. che acconsentono a riconoscersi mutuamente costituiscono la computational grid. Un utente che desidera utilizzare risorse di calcolo della computational grid deve appartenere ad almeno una V.O.; dagli amministratori della stessa viene fornito un certificato di identificazione univoco con il quale l'utente può, attraverso le User Interface preposte allo scopo, firmare un permesso di accesso alle risorse, sottomettere i job, monitorarne lo stato e riceverne il risultato. Al momento della sottomissione può decidere quali sono i requisiti software e hardware che il nodo di calcolo destinazione deve soddisfare, specificandoli attraverso il Job Description Language. Tutte queste richieste, assieme al job e dati di input, vengono consegnate ad un Resource Broker, ovvero un servizio che, in base alle specifiche dell'utente, seleziona il centro di calcolo migliore disponibile al momento, al quale delega l'esecuzione del job.

2.3 Middleware

Raramente centri di calcolo distinti utilizzano lo stesso insieme di hardware e software; si rende necessario introdurre un livello di astrazione che consenta di accedere alle risorse di calcolo tramite interfacce e protocolli standard.
In una parola: il Middleware. Ne è un esempio il Globus Toolkit riportato in [1], il quale racchiude al suo interno i servizi essenziali per poter realizzare

una computational grid.
Un'evoluzione del Globus Toolkit è gLite [2]: ogni centro di calcolo ha almeno un Computing Element che si occupa di dialogare con uno o più Resource Broker; job e dati in entrata/uscita vengono gestiti dagli Storage Element; i nodi di calcolo sono detti Worker Nodes. Ogni componente della computational grid è soggetta ad un assiduo monitoraggio del suo stato; anche l'ambiente software dei Worker Nodes viene monitorato, attraverso le V.O. di certificazione, che sottomettono job di test in modo automatico, ed avvertono chi di dovere nel caso di problemi.

2.4 Progetto LCG

Un esempio reale di computational grid è quella preposta all'elaborazione dei dati provenienti dall'ultimo acceleratore di particelle allestito presso il CERN a Ginevra, il Large Hadron Collider (LHC). La computational grid

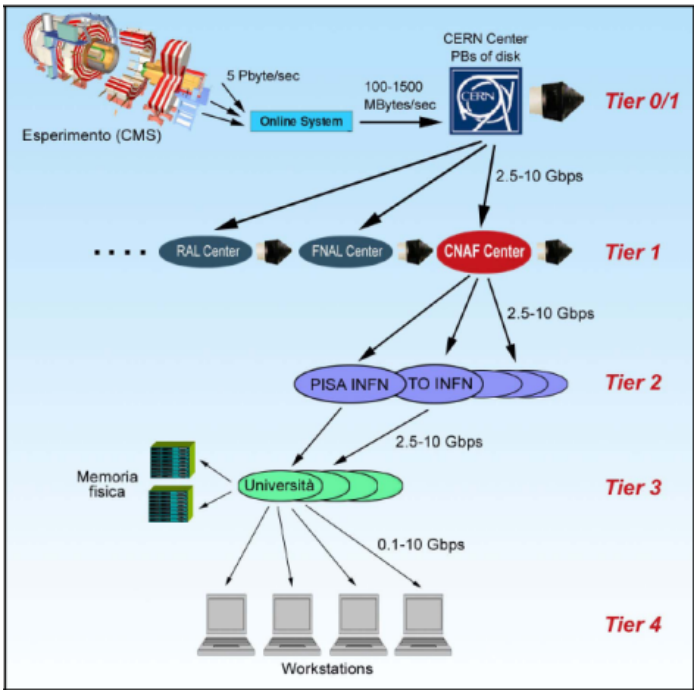

Figura 2.1: LCG computational data grid.

in questione è uno degli obiettivi del progetto LCG("LHC Computing Grid Project"). Le V.O. afferenti al progetto sono 4, denominate ALICE, ATLAS, CMS ed LHCb; ognuna di esse rappresenta un esperimento da effettuare con

LHC, con strumenti di rilevazione separati, ma in comune la richiesta di elevate risorse di calcolo. Queste vengono attinte dalla griglia computazionale, fisicamente distribuita in tutto il mondo.
Dai dati grezzi di ogni singolo esperimento, pari a qualche unità di PetaByte/sec, i sistemi inline ne selezionano per l'eleborazione solo una frazione, con la conseguente riduzione del throughput a qualche GigaByte/sec. Presso il CERN si trova un primo centro di calcolo, denominato Tier-0, il quale dispone di storage, soprattutto su nastro, per memorizzare i dati selezionati dai sistemi inline; dispone anche di risorse computazionali per una ricostruzione parziale dei dati. La successiva ricostruzione viene eseguita nei Tier-1 internazionali, collegati al Tier-0 con link a 2.5-10Gbit/s; l'unico Tier-1 presente in Italia è situato presso il CNAF dell'INFN a Bologna. Sia i dati ricostruiti che quelli grezzi rimangono nei sistemi di storage dei Tier-1. La successiva analisi è compito dei Tier-2 regionali; i risultati prodotti vengono archiviati al Tier-1 di riferimento, a disposizione degli altri centri.

2.5 Grid in Europa

Enabling Grids for E-sciencE (EGEE) è un progetto nato nel 2004 fondato dalla Commissione Europea per la costruzione di un'infrastruttura Grid a disposizione dei ricercatori 24 ore al giorno. Attualmente collega più di 250 istituzioni in 45 differenti stati del mondo; il middleware di riferimento è gLite, sviluppato aggregando vari progetti Grid, fra i quali Globus.

2.5.1 Grid in Italia

Nel 2002 arrivano i primi finanziamenti da parte del Ministero dell'Università e della Ricerca(MIUR) a sostegno del Consiglio Nazionale delle Ricerche(CNR) nell'ambito del Grid Computing.
A fine 2005 viene inaugurato a Bologna, presso il Centro Nazionale per la Ricerca e Sviluppo nelle Tecnologie Informatiche e Telematiche(CNAF) dell'Istituto Nazionale di Fisica Nucleare(INFN), il Tier-1 a supporto degli esperimenti LHC. Il CNAF inoltre opera attivamente nel campo delle computational grid, progettando e sviluppando parte del middleware gLite utilizzato in EGEE.
Una delle griglie è la "INFN Grid", che raggruppa tutti i centri di calcolo delle sedi distaccate INFN sul territorio nazionale; non è l'unico ente ad averne una: CNR, SPACI, ENEA, ICTP, INAF, INGV. Allo scopo di fornire una coordinazione nazionale per EGEE è nato il progetto Italian Grid Infrastructure(IGI), che raggruppa le sopracitate griglie.

Capitolo 3

Il caso CNAF

Per capire meglio come funziona un centro di calcolo di grandi dimensioni, consideriamo un caso reale.

3.1 La farm di calcolo

Come già accennato, il CNAF è 1 dei 10 Tier-1 a supporto degli esperimenti LHC; l'unico in Italia, attualmente dispone in totale di circa:

- 2K core,
- 4TB di RAM,
- 0.5PB di memoria di massa ed
- 1PB di storage su nastro.

3.1.1 Nodi di calcolo

I nodi fisici sono di proprietà dello stesso CNAF o in colocation da parte di sezioni INFN oppure centri di ricerca internazionali. Entro breve la maggior parte dei nodi verrà dotata di 2 cpu a 4 core con 16GB di RAM in totale; link Gigabit alla LAN interna su VLAN dedicata. Il sistema operativo standard è una distribuzione Linux, la Scientific Linux CERN (SLC) [3].

Constraint sulle risorse

Si è concordato, con le V.O. utilizzatrici della farm, la presenza di 2GB per ogni core di elaborazione.

3.1.2 Gestione dei nodi: QUATTOR

In una farm di tali dimensioni non è possibile dedicare troppo tempo a capire come risolvere un eventuale problema software, sia di sistema operativo che applicativo; dunque si preferisce reinstallare da zero il nodo incriminato. Bastano pochi comandi se si utilizza un sistema di gestione a profili:

- ad ogni nodo è associato un profilo di installazione, che rispecchia tutte le componenti software che devono essere presenti ed aggiornate con regolarità, oltre al set di configurazioni: dall'IP delle schede di rete fino alle partizioni dei dischi;
- al boot, ogni nodo chiede istruzioni sul da farsi, ovvero utilizza un meccanismo detto PXE, Pre eXecution Environment, con il quale è possibile sapere da un servente preposto se si può fare il boot da disco con la configurazione attuale, oppure se è necessario procedere ad una reinstallazione;
- come componente software, in ogni nodo vengono installati e configurati servizi daemon preposti al controllo di consistenza ed aggiornamento automatico;
- per controllo di consistenza si intende che il software installato in un nodo non può essere ne in difetto che in eccesso rispetto a quanto specificato nel profilo del nodo; in caso di divergenze, si provvede al ripristino.

Il gestore di profili utilizzato al CNAF è QUATTOR [4].

3.1.3 Il batch manager: LSF

Ogni Tier-X ha qualche grado di libertà nella scelta del batch manager, ovvero quella componente che si occupa di prendere in consegna dati e job da eseguire in un nodo, ritornando i risultati o un eventuale errore.
L'INFN ha stipulato una convenzione con Platform, la software house che produce LSF [5]. È un prodotto closed source, con possibilità di visionarne i sorgenti su richiesta; ecco le principali caratteristiche:

- ogni nodo viene identificato dalla sua configurazione hardware e dal sistema operativo, dagli slot ovvero il numero di job che possono essere eseguiti su di esso;
- per ogni nodo da gestire è possibile definire se i suoi slot sono dedicati in via esclusiva ad un determinato gruppo di utenti, oppure se fanno parte

dello shared cluster utilizzabile da chiunque, oppure una combinazione o nessuna delle due citate possibilità;

- l'accesso agli slot, di norma, prevede la sottomissione dei job ad una coda;
- la coda e scheduling con priorità e quant'altro viene gestito dal nodo master di turno, ovvero il nodo vincitore all'elezione distribuita alla quale partecipa ogni nodo;
- per ogni job che viene eseguito si collezionano le statistiche di utilizzo delle risorse, solitamente cputime ed occupazione RAM; queste statistiche servono per l'accounting.

3.1.4 V.O. e fair sharing

Ogni V.O. ha una propria coda, i suoi utenti sono autorizzati a sottomettere i job solamente su questa coda, ed in più si possono associare priorità differenti in base a singolo gruppo, utente o progetto interno alla V.O., con una politica di prioritizzazione gerarchica.
Si rammenta che ogni V.O. può avere a disposizione in via esclusiva un pool di nodi, ma anche accedere alle risorse condivise fra tutte le V.O.; un problema che si evidenzia è quello di evitare la monopolizzazione delle risorse nell'ambiente condiviso. Utilizzare una mera politica a priorità statiche non risolve il problema; si avverte la necessità di poter modificare le priorità in base all'effettivo utilizzo delle risorse. Ecco dunque il concetto di fair sharing:

- ad ogni V.O. viene assegnata una percentuale di utilizzo dello shared cluster, che il batch manager si impegnerà a far rispettare nell'arco di una finestra di ore, giorni oppure mesi;
- il fattore percentuale unito all'effettivo utilizzo contribuiscono a modificare in modo dinamico la priorità dei jobs in attesa nelle code; oppure si può pensare di mantenere statici questi valori di priorità, ma si rende necessario assegnare in modo dinamico il numero degli slot dello shared cluster che possono essere utilizzati da una V.O.;
- nei centri di calcolo di grosse dimensioni si preferisce usare la dinamicità del numero di slot; vi si trovano sempre dei job in attesa nelle code, quindi la percentuale di utilizzo non presenterà una varianza significativa nell'arco delle unità di tempo componenti la window di riferimento;

- nei centri di piccole dimensioni si preferisce usare la prioritizzazione dinamica; può capitare che un utente sottometta un numero considerevole di job in un periodo di calma, accaparrandosi tutta la potenza di calcolo disponibile ed ostacolando gli altri utenti, con una conseguente varianza marcata;

- le percentuali di fairshare possono essere a loro volta personalizzate all'interno di una V.O., costituento dunque un fairshare gerarchico, non ottenibile con parametri di scheduling statici;

- nel caso di prioritizzazione dinamica, la politica di fairshare può incentivare gli utilizzatori del batch system a rispettare in proprio le percentuali di utilizzo, proponendo incentivi da riscuotere o penali da pagare in base al comportamento tenuto; si rammenti sul fatto che l'intervento umano da parte dell'amministratore del batch system non si farà attendere a lungo.

3.1.5 Lo shared storage: GPFS

Finora si è parlato di computational grid, anche se nel caso della griglia LCG è più corretto parlare di computational data grid, per sottolineare la presenza massiccia della componente di archiviazione dei dati di esperimento e le successive rielaborazioni.
Tale mole di dati deve assere gestita con filesystem in grado di scalare senza troppi grattacapi; per un centro di calcolo la stabilità e scalabilità sono fattori irrinunciabili, ma da soli non bastano. Serve che ogni nodo possa accedere ad uno storage condiviso, senza single-point-of-failure e problemi di concorrenza. Si rende necessario introdurre il concetto di metafilesystem distribuito:

- separazione netta fra la componente storage fisica e quella che gestisce i metadati;

- ridondanza delle componenti per diminuire il rischio di perdita dei dati ed aumentare il throughput; meccanismo di file-locking distribuito;

- gateway hardware/software di accesso allo storage, con funzione di caching; feedback e cache-coerency distribuiti in scrittura;

- infrastruttura hotswap-on/off delle componenti gateway e storage, con limiti inferiori ben specificati per garantire la continuità del servizio;

- accesso allo storage da parte dei client tramite modulo software nel kernel per l'utilizzo dell'interfaccia di mascheramento VirtualFileSystem (VFS);

- gestione di volumi logici separati, con funzioni di checkpointing temporale;
- gestione logica di volumi simili, con funzioni di copy-on-write selettive;
- controllo degli accessi ai volumi;
- rimozione forzata di un nodo meta o storage nel caso di comportamenti anomali da parte dello stesso, anche tramite apparati hardware digitali di distribuzione della corrente elettrica.

Quanto detto sopra, purtroppo, rimane solo un concetto, poiche allo stato attuale non vi è un prodotto che soddisfi tutti i requisiti, se non a scapito della stabilità.
Una buona approssimazione viene fornita dal General Parallel File System (GPFS) [6] di IBM; utilizzato al CNAF, affiancato anche dall'Andrew File System(AFS). Nei centri di calcolo di piccole dimensioni si può trovare con facilità il Network File System(NFS), forse accompagnato da AFS; raramente GPFS, se non per testing.

3.2 La necessità

L'imperativo è contenere la complessità; se si può ridurla tanto meglio. Altro punto chiave è la security. Questi presupposti si adattano ad ogni tipo di progetto; nel caso specifico di una farm di calcolo con accesso Grid il personale umano dovrebbe gestire solamente situazioni di emergenza. Ogni job dovrebbe essere eseguito in un ambiente isolato(una sandbox), quantomeno logicamente, da quello di altri job presenti nello stesso nodo fisico. In aggiunta, la possibilità di customizzare l'environment di esecuzione è una richiesta sempre più frequente avanzata dalle V.O.(si parla di distribuzioni certificate da parte della V.O.), alle quali la griglia risponde dedicando un pool di nodi con accesso esclusivo configurati nel modo voluto.

3.3 La soluzione: virtualizzare

La separazione logica degli environment di esecuzione dei job può essere ottenuta tramite la virtualizzazione, introdotta nel capitolo 4.
Gestire più di una distribuzione certificata significa rispondere alle richieste degli utenti, ma anche introdurre non pochi problemi: aumenta la complessità di gestione; servono pesanti ottimizzazioni per mantenere la scalabilità necessaria a gestire milioni di job con i worker node on-demand. Tali problematiche si affrontano nel capitolo 6.

Capitolo 4

Tecniche di virtualizzazione

Una molteplicità di istanze di sistemi operativi attivi su di un unico calcolatore fisico, ma isolate l'una dalle altre come se fossero su calcolatori separati; ogni istanza gestisce autonomamente le proprie periferiche.
La suddetta separazione logica di istanze si può ottenere con la virtualizzazione; si assegnano in modo condiviso od esclusivo risorse del calcolatore(CPU, RAM, storage, network, ...) ad ogni istanza, la quale deve essere considerata come una blackbox; l'overhead introdotto per ottenere l'astrazione non deve compromettere le prestazioni. Le varie modalità di virtualizzazione, esistenti e future, possono venir meno ad alcuni degli aspetti anzidetti, per consentire ottimizzazioni dipendenti dal contesto.
Nel 1973 Gerald J. Popek e Robert P. Goldberg dettagliarono in [7] i requisiti formali che un'architettura di virtualizzazione deve necessariamente soddisfare.

4.1 Emulazione

Data una macchina astratta A, si può sempre simulare con essa il comportamento di un'altra B; è sufficiente codificare nel linguaggio di A un interprete per il linguaggio di B, anche se a volte risulta tutt'altro che facile. Per ricondurci alla virtualizzazione, si individua nel termine "emulatore" una componente software in grado di simulare un insieme non vuoto di macchine astratte; la simulazione può essere molto onerosa in termini di risorse computazionali.

4.2 Paravirtualizzazione

Se invece ci si accontenta di avere multiple istanze di sistemi operativi che possono essere eseguiti sulla medesima architettura, allora si sceglie un nodo di calcolo avente questa architettura e l'unica emulazione da fare sarà quella per trattare le istruzioni privilegiate ed eventualmente per l'accesso alle periferiche. Il vantaggio più evidente è l'eliminazione quasi totale dell'overhead che è necessario nell'emulazione di un'architettura differente da quella fisica del nodo.

4.3 Virtualizzazione a livello del sistema operativo

Nel caso l'obiettivo da raggiungere sia quello di avere tante istanze separate logicamente di un medesimo sistema operativo, allora si può ricorrere alla virtualizzazione a livello del sistema operativo. Prendendo ad esempio un sistema operativo Linux, a livello del kernel si troveranno replicate per il numero di istanze le tabelle dei processi, dei file aperti ... ovvero tutte le strutture che non interagiscono in modo diretto con l'hardware. Anche in questo caso le istruzioni privilegiate vengono intercettate e gestite dal software di virtualizzazione.

4.4 Supporto hardware alla virtualizzazione

Le tecniche finora viste consentono una virtualizzazione parziale dell'architettura hardware, ovvero non tutte le componenti sono accessibili direttamente da una virtual machine. La virtualizzazione totale si può ottenere se la stessa architettura è stata progettata allo scopo di supportare la virtualizzazione; già alla fine degli anni '60 alcuni mainframe della IBM furono progettati allo scopo.
Al giorno d'oggi questa tecnologia è alla portata di quasi tutte le tasche, dopo l'introduzione delle cosiddette "estensioni di virtualizzazione" nelle CPU consumer di AMD ed Intel.

4.4.1 Full virtualization

Mentre nella paravirtualizzazione è necessario apportare modifiche sostanziali al sistema operativo da istanziare, nella full virtualization ne sono eventualmente richieste di impatto quasi nullo.

Parte II

Studio di fattibilità

Capitolo 5

Virtualizzare un nodo di calcolo

5.1 Piattaforma di riferimento

Come accennato, la maggior parte delle V.O. ha già confezionato una propria distribuzione Linux di riferimento, customizzata secondo le loro esigenze. L'essere una distribuzione Linux apporta molti vantaggi, soprattutto se si intendono utilizzare, per convenienza, tecniche di virtualizzazione che richiedono modifiche al sistema operativo delle virtual machine.

5.2 Tool di virtualizzazione Xen

Di tool di virtualizzazione ve ne sono tanti, open o closed source, a pagamento o gratuiti.
Xen [8] è un tool opensource di virtualizzazione per ambiente Linux, realizzato dal gruppo "Networks and Operating Systems (NetOs)" del Computer Laboratory presso l'Università di Cambridge. Nato come paravirtualizzatore, ad oggi supporta la virtualizzazione in hardware(fully-virtualization). Recentemente il progetto è stato acquistato da Citrix Systems.

5.2.1 Dom0 e DomU: la struttura a domini

Nel tool Xen si distinguono due figure fondamentali:

- il dom0, ovvero il sistema operativo del nodo fisico;
- i domU, ovvero i domini logici che utilizzano le risorse del nodo fisico.

Il tool Xen viene anche detto "virtual machine monitor(VMM)" o "hypervisor"; i domU vengono gestiti dal dom0, il quale necessita di un Kernel

modificato, specifico per Xen; per i domU la modifica è necessaria se si sceglie di utilizzare la paravirtualizzazione.

5.2.2 Overbooking delle risorse

Prima di poter creare un nuovo domU bisogna accertarsi che vi sia a disposizione la quantità minima di RAM che vorremmo dedicarle ad uso esclusivo; non è ammesso l'overbooking della RAM, mentre per la CPU è possibile: ad esempio si consideri un quadcore e 6 istanze attive. Si può scegliere di assegnare 4 virtual CPU ad ogni istanza: questo significa che le istanze si contenderanno l'accesso ai core, e quale istanza accede a quali core viene deciso dallo scheduler delle cpu nel dom0.

5.2.3 La scelta della paravirtualizzazione

La paravirtualizzazione consente di gestire in modo più agevole alcuni aspetti della virtualizzazione, poichè si tratta di un astrazione più debole rispetto alla full-virtualization; un aspetto del quale non si è ancora accennato è la possibilità di avviare una virtual machine, fermarla e salvarne lo stato su storage, spostarla in un altro dom0 compatibile, ricaricarla in memoria e riattivarla. Inoltre è possibile effettuare tutte le fasi anzidette in real-time, ovvero si sceglie di migrare un'istanza da un dom0 ad un'altro, minimizzando il downtime.
Altro aspetto da considerare sono le performance delle virtual machine: ad esempio, si è verificato sperimentalmente che con i driver network nativi di un'istanza fully-virtualized(distribuzione Linux) si ottengono prestazioni decisamente scadenti rispetto ad utilizzare nella stessa istanza fully i driver di un dominio paravirtulizzato.
Considerando lo stadio di sviluppo attuale del tool Xen, si è scelto di utilizzare in seguito la paravirtualizzazione.

5.3 Installazione di un virtual environment

Se non si hanno a disposizione immagini preconfezionate per le istanze da avviare, necessariamente si deve procedere all'installazione delle stesse.
In un centro di calcolo di grandi dimensioni si trovano i gestori dei profili, dunque sarebbe gradito fare il boot PXE anche per una virtual machine. Utilizzando la paravirtualizzazione è necessario uno script esterno che verifichi il da farsi tramite PXE, dando le dovute indicazioni a Xen; di helper-script

ve ne sono tanti, un esempio in python è "pypxeboot" [9].
Il metodo classico via CD/DVD(o un'immagine dei supporti) è sempre valido, avendo l'accortezza di utilizzare un kernel modificato per domU Xen.

5.3.1 Le risorse di un domU

A seconda dei filesystem che si hanno a disposizione nel dom0 si possono fare ottimizzazioni; per iniziare, si consideri un filesystem journaled e niente più. Al fine dell'installazione si deve creare una partizione ad-hoc per ogni possibile istanza, oppure creare delle immagini della dimensione disco voluta.

5.3.2 Virtual block device

Seguendo lo stile introduttivo, si procede alla creazione di un'immagine utilizzando il comando **dd**:

Codice 1 *dd if=/dev/zero of=PATH bs=BLOCK_SIZE count=SIZE*
ove

- *PATH è il percorso contenente la destinazione dell'immagine da creare;*
- *BLOCK_SIZE esprime la dimensione in byte di un blocco, utilizzando le abbreviazioni K, M, G, T, P, E, Z, Y di ovvio significato(ad esempio, per 1 MegaByte si scrive 1M);*
- *SIZE è il numeri di blocchi che costituiranno l'immagine.*

Nulla vieta di creare più immagini, che verranno associate alla virtual machine come dischi fisici differenti; il numero è limitato dall'implementazione Xen, e si ricordi che ognuno di essi richiede un loop device in /dev/loop/(se non ve ne sono abbastanza, crearli).

5.3.3 Virtual CPU

Nei limiti del nodo fisico, si possono associare tante virtual CPU quante sono quelle del nodo; in ogni caso, almeno una.

5.3.4 Virtual network

Si possono associare più interfacce di rete, ognuna con il proprio MAC address; ogni interfaccia viene associata ad un bridge creato da Xen.

5.4 Profilo di un'istanza

La configurazione di un'istanza, ovvero il suo profilo, deve essere specificata a Xen in un file preposto, con la seguente sintatti:

Codice 2 *cat /xen/profile/wn-01-03-01-01-01*

```
name = "wn-01-03-01-01-xen-1"
memory = 2048
vcpus = 2
disk = [
        'file:/home/xen/img/bootDisk,hda,r',
        'file:/home/xen/img/wn-01-03-01-01-xen-1/swapCacheDisk,hdb,w',
        'file:/home/xen/img/wn-01-03-01-01-xen-1/writeDisk,hdd,w',
]
vif = [ 'mac=00:16:3e:00:aa:bb,bridge=xenbr1' ]
pae = 1
acpi = 1
bootloader="/usr/bin/pygrub"
```

ove

- *name: nome identificativo da associare all'istanza;*
- *memory: dimensione in MB della RAM;*
- *vcpus: numero delle virtual CPU;*
- *disk: lista ordinata dei block device; ordinata poichè il primo nella lista verrà considerato come il disco di boot;*
- *vif: lista delle virtual network interface; per ogni interfaccia si indicano MAC address e bridge al quale associarla;*
- *pae: se =1 viene abilitato il supporto all'estensione PAE nel domU;*
- *acpi: se =1 viene abilitato il supporto all'estensione ACPI nel domU;*
- *bootloader: path del bootloader da utilizzare.*

5.5 Tempi di avvio

Non si notano sostanziali differenze nei tempi di avvio di una virtual machine rispetto ad un nodo fisico; se invece si considera la possibilità di mantenere ibernate immagini già avviate, il restart dura il tempo di ripristinare l'immagine RAM del domU: con uno storage non troppo pretenzioso possono bastare qualche decina di secondi.

Capitolo 6

Worker node on-demand

Dopo aver appreso come si opera con le virtual machine, si risponda alla richiesta avanzata dalle V.O.: ad ogni job il suo sistema operativo; dal canto suo il centro di calcolo viene incontro all'esigenza, ma non intende modificare il codice del batch system e ridefinire la configurazione delle code. In più, non è ammessa la sottomissione di job nel batch system per gestire questi nodi di calcolo virtuali, poichè si contaminerebbero i dati di accounting.
A titolo riassuntivo, si fornisce un elenco delle richieste e dei constraint:

- per virtual environment si intende tutta la componente software dell'istanza di un nodo virtuale; il sistema operativo appartiene alla famiglia Linux;
- per ora, l'associazione (job, virtual environment) viene codificata in modo statico, ovvero le V.O. specificano le varie combinazioni di (utente, gruppo, progetto, virtual environment); in futuro, questa scelta potrà essere effettuata con una direttiva specifica in JDL al momento della sottomissione di un job;
- le virtual machine sono un'opportunità per arginare i problemi di sicurezza: si richiede che un'istanza sia associata ad un unico job in esecuzione; l'istanza deve essere distrutta al termine dell'esecuzione;
- la virtualizzazione non deve comportare una perdita eccessiva di performance;
- la gestione dei nodi virtuali non deve contaminare i dati di accounting, e tanto meno richiedere modifiche sostanziali ai sistemi con i quali deve interagire(batch system, shared storage, ...);

- esclusi i job di test, la maggioranza ha una durata prossima alle 10 ore, dunque la creazione e distruzione di un nodo di calcolo virtuale non è necessario che avvenga entro pochi secondi, ma qualche minuto è un ritardo più che tollerabile considerando la durata effettiva del job;
- la maggior parte dei nodi di calcolo fisici presenti nei centri di calcolo è dotata di proprio storage interno, oltre a quello shared; ovvero i nodi diskless sono la minoranza;
- ogni nodo di calcolo virtuale deve poter fornire tutti i servizi attualmente presenti sui nodi fisici; inoltre non è accettabile che i servizi del nodo virtuale siano impossibilitati a mantenere il sincronismo temporale;
- i virtual environment sono soggetti all'aggiornamento delle componenti: tali aggiornamenti non possono avvenire durante l'esecuzione di un job, ed inoltre non è auspicabile avviare nuove istanze con una versione obsoleta dell'environment;
- ogni nodo di calcolo virtuale deve avere almeno 1 core di esecuzione dedicato, 2GB di RAM e 7GB di spazio disco per l'utente del job, oltre a swap ed altre partizioni di servizio(ad esempio usate dai filesystem distribuiti per caching), un indirizzo IP pubblico e reverse lookup record registrato nel DNS; per ora non si considera la possibilità di dover gestire job MPI;
- l'utilizzo di nodi virtuali non deve comportare alcuna modifica da parte delle V.O.; deve risultare un'operazione trasparente.

6.1 Problemi

Leggendo la lunga lista di cui sopra di evidenziano numerosi problemi da gestire.

6.1.1 Molteplicità dei virtual environment

Ogni V.O. potrebbe voler definire più di un virtual environment(per brevità, in seguito v.e.); si assume che l'occupazione di ogni v.e. sia di qualche GB. Ogni dom0 può avere, nel caso pessimo, ogni istanza attiva con un v.e. differente dalle altre; ogni istanza necessita anche di altro spazio libero per l'utente e le partizioni di utilità.
A titolo di esempio, si considerino per ogni domU:

- 3GB virtual environment;
- 7GB spazio libero;
- 2GB swap;
- 1GB partizione di utilità.

Ovvero 13GB di spazio per ogni domU, diviso in 3 distinti virtual block device:

- hda 10GB;
- hdb 2GB;
- hdc 1GB.

Se si ha a disposizione un doppio quadcore, sono necessari almeno 91GB oltre allo spazio richiesto dal dom0 e cache dei v.e. da utilizzare; la cache serve a ridurre le richieste di retrieve.
Nel caso di nodi di calcolo diskless si intende nodi che non hanno alcun harddrive interno, ma accedono ad uno storage condiviso; l'accesso può essere consentito via Ethernet oppure iSCSI, FibreChannel o tecnologie similari. In questo caso si possono configurare aree di storage condivise, che consentono un risparmio di spazio e di costi, soprattutto nel caso di SAN(Storage Area Network) il quale hardware è piuttosto costoso. Con i diskless la cache può essere eliminata, poichè si ha un accesso diretto al pool di v.e. utilizzabili; in realtà, un caching generico viene comunque effettuato in modo trasparente a livello di architettura dello shared storage, per minimizzare l'accesso allo storage fisico e ridurre i tempi di attesa.
Fino ad ora si è parlato di "storage preposto alla memorizzazione dei v.e." senza specificare il soggetto, il quale dipende dal contesto; ognuno con propri limiti:

- un server web, con retrieve via HTTP; come sistema di caching: proxy;
- shared storage;
- supporto DVD per i nodi con lettore integrato;
- pendrive o memoria allo stato solido;
- ... (altro)

6.1.2 Allocazione MAC ed IP

Ogni nodo virtuale deve avere un MAC address univoco, al quale associare un indirizzo IP pubblico; si noti che il MAC address deve avere come codice vendor quello dedicato a Xen, ovvero della forma 00:16:3E:uv:wx:yz.
Per gestire ciò si evidenziano almeno due modi:

1. ad ogni dom0 vengono assegnati in modo statico un numero di MAC address univoci pari al numero di core usabili per i domU;

2. la creazione di un domU prevede la richiesta di rilascio(lease) di un MAC address ad un centro di distribuzione, con la restituzione nel momento della distruzione dell'istanza.

Rispettivamente, si evidenziano i seguenti vantaggi:

1. viene eplicitata l'associazione (IP/FQDN di un domU, dom0), rafforzando la security e la tracciabilità;

2. possibilità di creare globalmente immagini ibernate già avviate e configurate con MAC e IP disponibili al momento, pronte per essere migrate come domU sul dom0 richiedente.

Rispettivamente, si evidenziano i seguenti svantaggi:

1. impossibilità di avere immagini ibernate, ne non aumentando il numero di MAC address ed IP concessi ad un dom0; in ogni caso la produzione di un'immagine ibernata è specifica per un dom0, non utilizzabile da altri;

2. l'associazione (IP/FQDN di un domU, dom0) è dinamica; si richiede inoltre di comunicare con un centro di distribuzione esterno al nodo.

6.1.3 Pulizia di un environment usato

Se ad ogni domU si associa in esclusiva un core, allora il numero massimo di domU per un dom0 è il numero di core del nodo fisico meno uno(purchè vi siano 2GB di RAM per ogni core), quest'ultimo da dedicare al dom0, che deve essere usato per:

- gestire i domU;
- creare i file che ospiteranno i virtual block device dei domU;
- scaricare i v.e. e popolare le partizioni preposte;

- pulire le partizioni di un domU dopo l'uso;
- montare in loop i virtual block device di ogni domU attivo.

La "pulizia" riguarda i dati modificati dall'utente e dai demoni, facilmente ottenibile montando in loop la partizione, rimpiazzando i file presenti con una copia del v.e. della prossima istanza da avviare.
Si può pensare di prevedere ulteriori partizioni già popolate e pulite, da utilizzare per avviare istantaneamente l'istanza successiva, mentre in background si popola la partizione uscente con l'environment più probabile per il prossimo job; nei centri di calcolo di grandi dimensioni si ha il grande vantaggio di avere sempre dei job nelle code: basta osservarle per prevedere il prossimo environment.
La presenza di partizioni pronte all'uso aumenta l'occupazione disco; può accadere che la previsione dell'enviroment sia sbagliata, oppure che nella fase di clean si sia copiato un v.e. non aggiornato. La latenza maggiore si introduce nel caso non vi sia in locale nel dom0 una copia del v.e. necessario; il tempo per il retrieve è variabile, il minimo è pari alla larghezza di banda tra il dom0 e lo storage dei v.e., senza considerare il carico degli enti interessati.

6.1.4 Aggiornamenti periodici e demoni di gestione

Il gestore di profili per nodi di calcolo utilizza demoni installati su ogni nodo, allo scopo di verificare i parametri di configurazione e la composizione dell'environment: se si riscontrano differenze con il profilo assegnato, si provvede al ripristino con integrazioni e/o cancellazioni.
Nel caso di un nodo virtuale non sarebbe cosa gradita che, pochi istanti dopo il boot, la cpu rimanesse bloccata per svariati minuti allo scopo di effettuare i controlli di consistenza: è sufficiente che si sia utilizzata l'immagine di un v.e. aggiornata e certificata essere consistente.
Gli aggiornamenti, soprattutto quelli di security, avvengono con una ben determinata periodicità e vanno riflessi su ogni nodo interessato; meno modifiche ci sono agli environment più efficiente risulta l'operato del caching. La previsione di un aggiornamento ogni due settimane è piuttosto attendibile.

6.2 Ideazione

Ecco il ciclo di vita, ovvero tutto ciò che deve avvenire dall'arrivo di un job fino alla distruzione del nodo di calcolo on-demand:

- il job viene sottomesso al batch system; nel momento in cui diventa elegibile per il dispatch viene selezionato uno slot libero assegnato alla V.O. dell'utente che ha sottomesso il job;
- lo slot libero è associato ad un dom0, il quale deve avviare un domU configurato come un nodo di calcolo; quale v.e. utilizzare per il domU è un'informazione statica;
- avviata la virtual machine, si deve migrare il job su di essa; il job non deve essere eseguito su di un v.e. diverso da quello richiesto;
- un nodo virtuale è autorizzato ad eseguire solamente un job che è stato migrato in modo corretto secondo un rigido protocollo, ovvero deve esistere un meccanismo di concessione e verifica dell'autorizzazione;
- finita l'esecuzione, l'istanza del domU viene distrutta e si provvede al cleaning dell'environment.

6.2.1 Migrazione di un job

Nell'elenco di cui sopra si è utilizzato il concetto di migrazione di un job da un nodo ad un'altro; un concetto semplice da intuire, ma difficile da attuare senza provocare danni. I danni si verificano nel mancato rispetto dei protocolli di una delle componenti coinvolte nella gestione dei job, ovvero del batch system. Ad esempio, in LSF la migrazione di un job può avvenire solo se il job è di tipo "restartable", ovvero:

- è checkpointable, ovvero è possibile interrompere il job salvandone lo stato attuale, consentendo una volta migrato di essere rieseguito dal punto dell'interruzione. Il checkpointing può essere una caratteristica implicita del programma, ovvero dopo ogni elaborazione "interessante" vengono salvati in modo incrementale i risultati, e al restart il job continua l'esecuzione dal punto successivo all'ultima delle elaborazioni "interessanti"; la presenza esplicita, invece, corrisponde ad una gestione accurata del segnale che viene inviato al processo "main" associato al job;
- oppure non è importante(non ci sono effetti collaterali) se il job debba essere rieseguito dell'inizio(con o senza checkpointing implicito).

Si conviene che, se il job non viene eseguito fintanto che non si trova nel proprio v.e. in un domU preposto, non vi sono problemi.

6.3 Ottimizzazione

Si avverte la necessità di codificare un modulo software che si occupi di prendere in consegna il job, creare un nodo di calcolo virtuale con il v.e. corretto, e quant'altro; il capitolo 8 tratta ciò, sotto il nome di virtual manager. Il protocollo che il gestore seguirà dipende indissolubilmente dalle ottimizzazioni che vengono attuate per gestire al meglio il servizio di nodi di calcolo virtuali on-demand; nel proseguo alcune delle ottimizzazioni possibili, focalizzate sulla componente storage.

6.3.1 Caching

Come ampiamente accennato, un job deve essere eseguito nel suo environment di riferimento, che nel peggiore dei casi può avere una dimensione di qualche GB; se il dom0 non ha a disposizione l'immagine del v.e. deve procurarsela dallo storage dei v.e.: nei centri di calcoli di piccole dimensioni le reti Gigabit non sono così diffuse come si potrebbe pensare, dunque fare il download di un'immagine impiega svariati minuti; si spera che nessuno osi cancellare l'immagine ottenuta alla distruzione del domU associato. Ma potrebbe capitare che non vi è altro modo, poichè non c'è abbastanza spazio disco per più di un environment in cache. Il rischio che si corre è di avere un nodo "lento" allo stato pratico, ovvero in "trashing", a maggior ragione se i job da eseguire sono di certificazione. Il trashing si può evitare se il batch system riesce a gestire il fairsharing non a slot generici, ma a slot su di un nodo con lo stesso v.e.: una volta fatto il retrieve dell'environment, esso viene usato per tutti i domU da avviare nei giorni seguenti, fino ad un aggiornamento; anche più di un v.e. per nodo fisico è accettabile, purchè non si verifichi una varianza marcata, considerando una window di qualche giorno, degli avvicendamenti dei v.e. nella cache del nodo in questione.

6.3.2 Read-only su shared storage

Poichè solo una frazione dell'environment verrà effettivamente utilizzata per l'esecuzione di un job, sarebbe cosa gradita evitare il retrieve completo. Chiedere all'utente di specificare tutte le dipendenze delle quali il suo job necessita potrebbe non essere soddisfacente, poichè oltre a privare l'utente di qualche secondo della sua vita, alcuni utenti non conoscono il significato di "dipendenza" e non hanno intenzione di capacitarsene nel breve termine. Le dipendenze di un file compilato nel senso delle librerie richieste non copre tutte le possibili forme nelle quali si manifestano i job, poichè sono spesso presenti, ad esempio, elementi di scripting.

Dato che non si è in grado di prevedere tutte le dipendenze che occorreranno durante l'esecuzione del job, si può scegliere di attendere il momento in cui si manifesta la richiesta ed eseguire il retrieve di ciò che serve, ovvero:

- si codificano dei v.e. minimali, che consentano di fare il boot e montare un filesystem di rete dove si trova tutto il resto del v.e.; il filesystem può anche essere esportato dallo stesso dom0 verso i suoi domU;
- oppure si può scegliere di partizionare il v.e. in due, separando le componenti read-only(r.o.) da quelle read/write. La parte r.o. può essere codificata come partizione a sola lettura, passata come virtual block device ad un'istanza da creare; si può scegliere se memorizzarla solamente sullo shared storage, oppure averla anche in cache localmente ad un nodo se è previsto un uso intensivo di tale environment.

Si noti che il primo approccio è basato su file, mentre il secondo su blocchi di dati. In entrambe i casi è necessario stimare la latenza introdotta e verificare se la scalabilità del sistema viene preservata dalla scelta effettuata; inoltre in caso di aggiornamenti bisogna prevedere di mantenere attiva anche l'immagine r.o. attualmente presente fintanto che vi sono domU che la usano. Nel caso di nodi diskless lowcost(nessun HBA speciale per l'accesso allo storage) l'approccio da adottare non potrà essere troppo distante da quanto illustrato, a meno che si scelga di utilizzare dei ramdisk per sopperire alla mancanza di storage(se si è pensato per un momento di utilizzare quest'ultima soluzione, si consiglia vivamente di acquistare un harddrive).

6.3.3 Prefetch degli environment osservando la coda dei job

Per ridurre i tempi di avvio si può pensare di guardare quali sono i prossimi job che dovranno essere eseguiti e inizializzare tutto il necessario per ottenere l'avvio dei domU in tempi ridotti; purchè tutto ciò non sia a sfavore dei job attualmente running: non è ammissibile che vengano fatti "rallentare" a favore dei posteri.

6.3.4 Il parcheggio delle macchine ibernate

In ambiente SAN e nodi diskless, con performance significative nell'accesso e utilizzo dello shared storage, si può estendere l'idea del prefetch creando un "parcheggio" autoalimentantesi:

- inizializzazione:
  - si pensi alla riapertura delle code dopo un'operazione di draining delle stesse;
  - arriva un job e si creano tutte le partizioni necessarie per avviare un domU, oltre ad inizializzare il v.e. corretto;
  - si avvia tale domU, si attende la terminazione del job e si distrugge l'istanza di nodo virtuale;
  - si effettua il clean delle partizioni e l'eventuale aggiornamento, si fa ripartire il domU e lo si iberna, salvando il risultato nel parcheggio, un'area di storage dedicata ai nodi di calcolo virtuali ibernati;
- fase operativa:
  - all'arrivo di un job si verifica se è presente nel parcheggio un domU ibernato con il v.e. richiesto: se sì, il dom0 acquisisce la proprietà dell'immagine ed avvia il nodo virtuale; se no, si provvede alla costruzione ed inizializzazione del necessario;
  - terminato il job, si verifica le presenza di un aggiornamento per il v.e. corrente ed eventualmente si effettuano le modifiche; si fa ripartire il domU, se ne fa l'ibernazione e si rilascia il lock su di esso;
  - se si è scelto di limitare il numero di immagini ibernate presenti nel parcheggio, allora si deve fare in modo che il parcheggio rispecchi le necessità dei futuri job, attualmente in attesa nelle code;
- aggiornamenti:
  - all'avvento di un aggiornamento le immagini ibernate non allineate devono essere invalidate;
  - si può scegliere di cancellare le immagini invalidate, oppure incaricare un dom0 per l'aggiornamento e la reimmissione nel parcheggio.

Si può prevedere di dedicare alcuni nodi al popolamento e mantenimento del parcheggio, nel caso lo si ritenga utile.

6.3.5 Shared home sul Dom0 via NFS

Sui 13GB previsti per ogni istanza di un nodo di calcolo virtuale, ben 7GB pari a più del 53% dello spazio è richiesto essere disponibile per l'utente. Si

può decidere di risparmiare un po' di spazio facendo si che il dom0 esporti ai propri domU via NFS quantomeno lo spazio che gli utenti possono usare, purchè si possa mantenere separati logicamente utenti di domU differenti.

6.3.6 Volumi logici con copy-on-write selettivo a blocchi di dimensione variabile

Nel caso di nodi diskless, oppure nodi con elevato numero di core di elaborazione, lo spreco di spazio per memorizzare gli environment potrebbe far lievitare i costi e scoraggiare l'adozione della virtualizzazione. Per poter risparmiare, in tutti i sensi, si può pensare anche ad una virtualizzazione dello storage, con la creazione di volumi logici:

- un volume è l'insieme di storage fisico e metadati di un sottoalbero di un filesystem;
- se più utenti devono accedere agli stessi identici dati, ma con la richiesta che un utente rimanga isolato dagli altri, allora si possono creare tante copie logiche di un volume "madre" quanti sono gli utenti;
- le modifiche di un utente ai dati o alla struttura del filesystem si riflettono solamente sul suo volume logico;
- non è necessario memorizzare più di una volta un file se è presente in più di un volume logico; si utilizzano i metadati per tenere traccia di ciò;
- non è necessario allocare interamente un file se un utente ha fatto una modifica locale(non globale) rispetto alla copia madre; basta tenere traccia dei dettagli della modifica, sempre con metadati appositamente codificati.

6.3.7 Partizione ad espansione dinamica

Se si ritiene che una gestione logica dei volumi sia più complicata che utile, allora si può ricorrere, per i dati degli utenti, alle partizioni ad espansione dinamica: non si occupano inutilmente blocchi di memorizzazione.

6.4 La soluzione più elegante/ideale

Le ottimizzazioni proposte sono fortemente legate al contesto applicativo; ecco dunque un resoconto della soluzione migliore che si può scegliere di adottare in base alle proprie risorse:

- nodi di calcolo "datati", almeno dualcore, scheda di rete 10/100, qualche decina di GB di storage: si può dedicare il nodo in via esclusiva ad una sola V.O. oppure shared ma con un numero limitato di v.e.. Se il numero di core è limitato non si avvertono benefici nell'utilizzo di domini logici: lo shared home sul dom0 via NFS può rivelarsi invece una scelta obbligata. Con l'upgrade dell'interfaccia di rete ad 1Gb/s si può prendere in considerazione l'uso di domU ibernati e rilassare il vincolo di gestione di pochi v.e.;

- nodi di calcolo "moderni" con un numero di core modesto, storage:

  - interno a basse prestazioni: si può adottare la strategia read/write con supporto dello shared storage via rete, oppure ad una gestione totalmente locale; si rende necessario popolare e fare il clean delle partizioni anche quando è attivo il massimo numero di domU;
  - interno ad alte prestazioni: come per il punto precedente, ma si può demandare l'esecuzione delle operazioni sulle partizioni al momento del bisogno;
  - su SAN via HBA dedicato: l'approccio da adottare è quello dei volumi logici;
  - interno e SAN: si utilizza lo storage interno come cache e per i dati degli utenti; fintanto che vi è spazio si utilizza lo storage interno;

- nodi di calcolo con elevato numero di core: quando il numero di core cresce e non si utilizzano job MPI o simili, allora il throughput richiesto da processi io-bounded può diventare difficile da soddisfare se il nodo è in configurazione diskless, a meno che si scelga di dotarsi di un'infrastruttura di rete Infiniband: oltre ad essere una scelta costosa, può non bastare ad arginare i problemi di latenza e performance. Ricorrere ad uno storage "tampone" interno ad alte prestazioni diventa una scelta quasi obbligata.
  Nodi ad 80 core non sono così lontani, ma rimangono utili solamente con job distribuiti, ove solo la minoranza dei processi è io-bounded verso lo storage.

Capitolo 7

Centri di calcolo di piccole dimensioni

Poichè la virtualizzazione è alla portata di quasi tutte le tasche, anche i centri di calcolo di piccole dimensioni possono pensare di adottarla; per i centri che devono ancora nascere si consiglia di adottarla fin dal principio, consentendo di scegliere in anticipo le ottimizzazioni da effettuare, potendo così risparmiare sui costi dell'hardware.

7.1 Il batch manager: Torque-MAUI

Un batch manager come LSF potrebbe essere troppo complicato da gestire; inoltre, per pochi worker node non è nemmeno necessario. Torque-MAUI è l'alternativa preferita, grazie alla presenza massiccia di howto e configurazioni pubblicate da altri centri di calcolo sui loro wiki [10]; inoltre sono presenti nel middleware le interfacce necessarie per l'utilizzo in Grid.
In breve le potenzialità:

- code multiple, con possibilità di definire priorità e fairsharing;
- non supporta il fairsharing gerarchico;
- supporto alla sospensione, ma non alla migrazione in modo altrettanto esplicito;
- possibilità di effettuare reservation sulle risorse di calcolo, ovvero dedicare un pool di risorse ad un certo numero di utenti;
- possibilità di definire dei tag identificativi da associare ad ogni nodo, per specificare ad esempio il sistema operativo, la tecnologia di interconnessione con altri noti, etc.;

- non si possono impostare script di pre/post esecuzione.

7.2 Worker node on-demand con Torque

Torque non è stato progettato con lo scopo di supportare la virtualizzazione, dunque bisogna trovare una soluzione che sia facile da gestire; inoltre non vi è alcun supporto all'aggiunta di nodi on-demand: per poterne aggiungere bisogna editare il file di configurazione dei nodi e riavviare il servizio. Ecco dunque una possibile strategia:

- per ogni nodo da virtualizzare si definisce il numero massimo di domU che potranno essere avviati su di esso;
- ad ogni domU dovra essere assegnato un MAC address univoco ed un IP; poichè il centro di calcolo è di piccole dimensioni, si può decidere o meno se assegnare IP su rete nascosta. La differenza sostanziale è che i singoli nodi non potranno essere monitorati dai servizi di Grid preposti se l'IP è su rete nascosta; l'eventuale retrieve di dati da extranet viene concesso attraverso NAT/PAT o proxy;
- i nodi non sono accessibili in modo diretto dai job sottomessi, ma un job deve essere autorizzato "manualmente" ad essere eseguito su di un particolare domU;
- date le dimensioni del centro, potrebbero capitare che il sistema sia scarico: sarebbe cosa gradita accendere i nodi fisici nel momento del bisogno, con dispositivi hardware dedicati al controllo dell'alimentazione elettrica dei nodi; idem spegnerli quando non servono. Però il numero di slot disponibili che "vede" il resourse broker deve comprendere anche quelli dei nodi spenti.

7.2.1 Resource reservation

Con il meccanismo di resource reservation si richiede allo scheduler di riservare un certo numero di slot per un periodo di tempo, limitato o fino a revoca; questo stratagemma serve a:

- far "vedere" al resource broker gli slot liberi; ogni slot corrisponde ad un domU;
- evitare che un job vada in esecuzione su di un domU con v.e. diverso da quello richiesto.

Dunque, all'arrivo di un job si ha che:

- il job rimane in stato di wait poichè non vi sono risorse disponibili direttamente accessibili;
- un servizio si occupa di verificare i job in coda e trovare il v.e. richiesto dall'utente del job;
- se nel pool di domU riservati vi è un domU corrispondente al v.e., allora il job viene forzato ad essere eseguito sul domU in questione; altrimenti viene selezionato un domU per l'ibernazione, oppure per la distruzione. Al suo posto si crea/sveglia un domU con v.e. corretto;
- alla fine del job, il domU viene distrutto e si effettua la fase di clean, etc..

Si noti che si è assunto di avere già tutti i domU attivi, magari con v.e. differenti presenti in numero pari alle statistiche di utilizzo del periodo. La strategia di ottimizzazione ideale da adottare è quella delle immagini ibernate, ma solo se si ha a disposizione uno shared storage performante, altrimenti si ricorre al solito metodo di creazione e distruzione affiancato da una shared home via NFS.

7.2.2 Job labeling

Si può anche decidere di non prevedere la tecnica di virtualizzazione per ogni nodo di calcolo del centro; con la scelta di riservare le risorse non vi è modo che un job inizi l'esecuzione su di un nodo con v.e. differente da quello richiesto per il job.
Invece che forzare esplicitamente, a livello del batch manager, un job in arrivo ad essere eseguito su di un particolare domU, si può adottare un metodo più "gentile" che apporti maggiore tracciabilità:

- le varie richieste hardware e software, fatte da un utente nel JDL associato ad un job, vengono fatte presenti al batch system con delle direttive apposite, oppure aggiungendo alla descrizione del job dei tag/label identificativi delle scelte;
- uno di questi tag potrebbe essere utilizzato per specificare il v.e. di destinazione;
- tale specifica del tag verrebbe fatta dal servizio preposto al controllo dei job in coda, il quale farebbe un'operazione di job labeling;

- sarà il batch system a selezionare il domU di destinazione;
- nei log del batch system sarà codificato in modo chiaro quale era l'environment richiesto dal job.

Ovviamente può capitare che fra i domU attivi non ve ne sia uno con il v.e. adatto: il servizio deve accorgersi che il job è bloccato a causa di mancanza di risorse, provvedendo a riguardo.

Parte III

Sviluppo

Capitolo 8

Il virtual manager

Ecco nel dettaglio che cosa avviene dopo il dispatch di un job allo slot destinazione, nel caso del batch manager LSF, in assenza di virtualizzazione:

- l'eseguibile ed i dati di input vengono consegnati al demone LSF locale al nodo fisico nel quale si trova lo slot destinazione;
- viene eseguito uno script, il pre-exec, il quale viene utilizzato per controlli amministrativi ed inizializzazione dell'environment;
- si esegue un altro script, il job-starter; sole se tale script termina positivamente allora viene eseguito il job vero e proprio;
- alla terminazione del job, viene eseguito un altro script, il post-exec.

Tutti gli script sopra citati, ovvero pre/post-exec e job-starter, vengono eseguiti con i privilegi dell'utente che ha sottomesso il job; ogni V.O. può personalizzare pre/post-exec mentre il centro di calcolo si riserva il job-starter. Nel caso della virtualizzazione, si dovrebbero prevedere le seguenti fasi:

- il job utente arriva sul dom0 che possiede lo slot associato al job;
- deve essere predisposto tutto il necessario per avviare un domU con l'environment associato a (coda, utente, gruppo, progetto);
- se lo slot virtuale è l'ultimo slot reale disponibile nel dom0, allo si deve procedere alla chiusura amministrativa del dom0;
- rilascio dell'autorizzazione all'avvio del job sul domU specificato; creazione del domU;
- richiesta ad LSF di migrazione del job verso il domU appena creato;

- al termine del job distruzione del domU e cleaning, rilascio del lock sullo slot ed eventualmente riapertura del dom0.

I privilegi necessari per gestire le virtual machine ed il dom0 nei confronti di Xen ed LSF equivalgono allo status di amministratore: root nel caso di Xen, Administrator(o root) per LSF. Si intuisce che non è possibile concedere all'utente stesso questi privilegi, con il rischio di un'esplosione dei problemi di sicurezza; due delle possibili soluzioni sono le seguenti:

- l'utente sottopone la richiesta di creazione di un domU, con il v.e. destinazione, ad un servizio preposto; stessa richiesta per la distruzione dello stesso domU al termine dell'esecuzione del job;

- l'utente si pone in stato di wait, fintanto che il suo job viene migrato in un domU con il v.e. richiesto; a questo punto può partire il job. Al termine l'utente non deve fare alcunché: ci pensarà qualcun'altro alla distuzione del domU.

Dei due approcci presentati, il secondo è quello migliore dal punto di vista della security: l'utente non interagisce con alcun servizio di gestione della virtualizzazione; inoltre non può influire sulle decisioni che tale servizio deve prendere.

8.1 Demone o cronjob?

Prima di affrontare la codifica di un gestore della virtualizzazione è il caso di decidere se avrà le sembianze di un demone oppure di un semplice cronjob; vantaggi e svantaggi:

- un cronjob viene eseguito a distanza di qualche minuto; nel caso si verifichino situazioni di errore, esse perdurerebbero fino al nuovo avvio dello script di gestione; impostare lo stesso intervallo di avvio dello script su tutti i nodi non è desiderabile: il batch manager potrebbe andare in trashing con tutte le richieste che si troverebbe a gestire;

- un demone su ogni nodo controlla in neartime lo stato dei job e dei domU, può gestire gli errori appena si verificano ed essere più reattivo in caso di terminazione di un job.

8.1.1 Unico servente o approccio distribuito?

L'approccio ideale è quello semidistribuito, ma indebolisce i criteri di security che ci si è posti:

- si suddivida lo shared cluster in gruppi di qualche decina di nodi;
- ogni gruppo gestisce in modo autonomo i propri domU distribuiti sui vari nodi del gruppo;
- inoltre, ogni gruppo gestisce un'area di storage distribuita fra i suoi nodi, oltre che ad una cache distribuita per i v.e.;
- ogni gruppo ha il proprio parcheggio di immagini ibernate.

8.2 Implementazione

Si è scelto di implementare un demone da installare su ogni nodo fisico, indipendentemente dal fatto che il nodo possa o meno ospitare dei domU.

8.2.1 Xen ed i privilegi

Come ampiamente accennato, i privilegi necessari per operare con le virtual machine sono molto elevati, pari a quelli di root. Si può in ogni caso utilizzare meccanismi che concedono privilegi solamente ad alcuni utenti e per ben determinati eseguibili, ma in ogni caso l'utente, che ha sottomesso il job e passerà da almeno un dom0 ad un domU, non deve poter essere in grado di compromettere la sicurezza del centro di calcolo:

- in ogni dom0 viene creato un nuovo utente, adibito all'esecuzione del demone di gestione della virtualizzazione;
- i privilegi concessi sono solo quelli strettamente necessari al funzionamento del demone e niente più;
- a nessun altro utente vengono concessi tali privilegi, ad esclusione dell'utente root.

8.2.2 Real slot e virtual slot

Ogni core usabile del dom0 costituisce uno slot "reale", mentre gli slot "virtuali" costituiscono i v.e. cleaned pronti ad essere utilizzati, ovvero quei v.e. in cache: il numero degli slot virtuali è almeno pari a quello dei reali; per ottenere performance migliori è necessario che gli slot virtuali superino in numero quelli reali.

8.2.3 Gestione dei virtual environment

La soluzione più semplice prevede la creazione dei domU con tutte le partizioni necessarie in locale:

- si effettui la chiusura amministrativa del nodo;
- si verifichi la presenza di tutte le componenti software necessarie alla virtualizzazione: in caso contrario, apertura del nodo e shutdown del demone;
- si calcoli il numero di core usabili;
- per ogni core usabile si verifichi la presenza delle partizioni necessarie per i v.e. e quelle di utilità; in mancanza di parte di esse si provveda a crearle;
- si calcoli il numero di slot virtuali che possono essere tenuti in standby, pronti all'uso; si creino le necessarie partizioni per ogni slot;
- si verifichi se i v.e. contenuti nella cache locale necessitano di un aggiornamento; se sì, lo si effettui;
- verificare se le partizioni già popolate con dei v.e. siano aggiornate; provvedere all'aggiornamento se l'esito è negativo;
- apertura del nodo;
- fase di gestione:
  - si verifichi di frequente se vi siano dei job arrivati sul dom0: se vi è uno slot reale libero, allora predisporne uno virtuale con il v.e. desiderato ed avviare il domU. Migrare il job sul domU è provvedere ad un'eventuale chiusura amministrativa del nodo nel caso non vi siano più slot reali disponibili: se la chiusura viene effettuata e vi sono ulteriori job nel dom0 oltre a quello migrato, provvedere al riaccodamento degli stessi;
  - si verifichi di frequente se vi siano job terminati, sia in modo corretto che a causa di un errore: si provveda alla distruzione del domU associato al job; cleaning dell'environment e rimozione del lock sullo slot virtuale; contestualmente si provveda a liberare lo slot reale usato per il domU.

- nel mentre della fase di gestione le partizioni dei domU distrutti, associate ad uno slot virtuale, possono essere conservate o adattate in background ai v.e. richiesti dai futuri job; il tutto nei limiti dello storage interno.

Il caching degli environment è un requisito fondamentale: si rinunci a qualche slot virtuale a favore dell'immagine di un v.e. che si prevede verrà effettivamente utilizzata in seguito.

8.2.4 Clean degli environment usati

La soluzione che si è adottata, ovvero un insieme di partizioni associate ad ogni domU, prevede di effettuare il clean dei dati modificati dal sistema operativo o dall'utente durante il periodo di vita della macchina virtuale. La pulitura si può effettuare in vari modi; ad esempio:

- formattare la partizione con il filesystem desiderato;
- montare in loop e popolare con i file necessari.

Questa operazione di cleaning può essere effettuata a bassa priorità anche in background, per ridurre il tempo di avvio di un nuovo domU; ciò si può fare sole se si ha a disposizione uno slot virtuale libero da associare ad un eventuale nuovo job in attesa sul dom0.

8.3 Coordinazione distribuita per l'interazione col batch system

Nel caso di centri di calcolo di piccole dimensioni non si avverte la necessità di porre un limite al numero di richieste informative fatte al batch system, ad esempio per verificare i job in coda o lo stato di un job: i nodi sono talmente pochi che non si avvertono rallentamenti significativi nei tempi di risposta da parte del sistema di gestione. Al contrario, in un centro di grandi dimensioni il rallentamento si avverte eccome; si deve provvedere a porre un limite alle richieste. Ad esempio:

- in presenza di cronjob, l'accesso alle informazioni deve essere temporizzato: ad ogni gruppo di nodi viene assegnata una finestra di tempo nel quale si garantisce un tempo di risposta limitato superiormente; si osservi che vari cronjob possono essere preesistenti alla scelta di utilizzare un gestore della virtualizzazione;

- al posto di una finestra statica, si può predisporre un sistema per la prenotazione di una finestra;
- oppure si può scegliere di assegnare le richieste ad un proxy: il client potrebbe anche essere disposto ad avere informazioni in cache e non aggiornate all'ultimo istante.

Il rallentamento nei grandi centri, per la verità, potrebbe anche essere causato da un collo di bottiglia insito nello stesso batch system: mantenere i log di ogni job, fairshare e code su file distinti destrutturati, invece che record di database, di certo non promette bene:

- se il batch system prevede la possibilità di utilizzare un database, apportare le necessarie modifiche ai file di configurazione; inoltre, se lo si ritiene utile, popolare il database con i log storici;
- altrimenti, ne non si può intervenire sul codice ed interfacce del batch system, allora provvedere ad allestire un servizio di proxy tale che:
  - i demoni propri del batch system, distribuiti sui nodi di calcolo, siano alpiù le uniche entità autorizzate ad interrogare direttamente il batch system;
  - tutte le altre entità pongono le loro richieste al sistema di proxy, il quale può fornire sia informazioni up-to-date che non sincronizzate a seconda della sensibilità alla latenza e criticità del servizio che ha posto la query.

Si noti che la seconda possibilità di cui sopra non ha i requisiti di scalabilità richiesti, poichè se non si può accedere al codice del batch system, non si può far si, ad esempio, che i demoni di LSF presenti su ogni nodo utilizzino il sistema di proxy.

Capitolo 9

Test su scala

I centri di calcolo, indipendentemente dalla dimensione, non vedono di buon occhio una modifica radicale delle proprie modalità operative e configurazione dei servizi; dunque i primi test da eseguire non possono avere troppe pretese, e tantomeno compromettere la stabilità del centro.

9.1 Piattaforma

Il CNAF ha messo a disposizione un rack con una quarantina di nodi fisici dualcore, 2GB di RAM, storage interno e rete a 100Mb/s, uplink 2Gb/s verso la rete interna. Anche se viene meno ai requisiti di 2GB a core, per i primi test il vincolo può essere rilassato; inoltre il numero di real slot considerato sarà 2 invece che 1. Per l'occasione sarebbe cosa gradita provare anche la nuova versione di LSF, che inizia ad introdurre il supporto alla virtualizzazione, con i demoni codificati ad-hoc per le virtual machine. L'installazione dei nodi avviene configurando un profilo per il gestore dei nodi, e con pochi comandi e reboot il tutto è fatto.

9.1.1 Virtual environment

Per iniziare può bastare un numero di environment ristretto, e per abbreviare i tempi di deployment si può scegliere di rinominare un unico environment madre piuttosto che installarne vari differenti.

9.1.2 Storage domU

Ogni domU avrà in locale nel nodo fisico le partizioni per v.e. ed utilità; nessuna particolare ottimizzazione, a parte il caching ed eventualmente la shared home via NFS.

9.2 Retrieve immagini dei V.E. via HTTP

Il modo più semplice per il retrieve dei v.e. è quello di utilizzare il protocollo HTTP:

- l'uplink di 2Gb/s verso la rete interna viene impiegato, fra le altre cause, al retrieve dei v.e. da un server http; non è così difficile che tale server sia preesistente, come nel caso del CNAF;
- il server web deve essere ben collegato alla rete, con varie interfacce gigabit o addirittura Infiniband;
- per ridondanza e bilanciamento del carico, possono essere previsti più di un server web;
- allestire un sistema di caching con semplici software proxy per minimizzare il transito dei dati ed il carico degli elementi coinvolti;
- si può pensare di configurare i nodi di calcolo come sistema di cache distribuita.

9.3 Risultati attesi

Prima di eseguire il test è importante chiedersi quali saranno i risultati ed individuare i punti critici che potranno creare qualche problema:

- i nodi hanno un collegamento alla rete di basso throughput; nel caso il v.e. richiesto da un job non si trovi nella cache del dom0, il tempo per l'avvio del domU correlato aumenterà di almeno 30 secondi in condizioni ideali;
- la scelta di avviare 2 domU su di un nodo biprocessore, invece che dedicare un core alla sola gestione delle virtual machine, potrebbe portare a qualche rallentamento;
- dato l'impiego di solo 1GB a core, alcuni job potrebbero fare un uso intensivo dello swap, con un calo delle prestazioni rispetto all'esecuzione in un nodo di calcolo standard.

Per il resto non ci si aspettano problemi rilevanti al fine dell'impiego su scala.

9.4 Analisi dei risultati

A causa della nuova versione di LSF da testare non è stato possibile completare i test di scalabilità; nonostante la versione sia stata progettata appositamente per il supporto delle virtual machine, l'installazione del software sul worker node domU falliva, al contrario della vecchia release. Il supporto tecnico della software house ha risposto con un "la prossima release sarà pienamente compatibile"; si rimane in attesa.

Capitolo 10

Conclusioni

Sia la virtualizzazione che Grid sono la realizzazione di anni di studi e speranze, ideologiche e filosofiche; ma il bello deve ancora arrivare. I nodi di calcolo virtuali on-demand sono un esempio delle applicazioni possibili, un pattern per l'infrastruttura di utility computing, o cloud computing come lo ha rinominato l'IBM.

10.1 Sviluppi futuri

La sfida è iniziata da pochi anni, dunque il lavoro da fare è parecchio:

- fare dei test significativi;
- realizzare un'infrastruttura a basso costo ed alte prestazioni, con la componente storage che la fa da padrone: un metafilesystem distribuito che rasenti lo stato dell'arte;
- supporto ai job MPI;
- direttive di specifica dell'environment desiderato nel JDL, con tutte le conseguenze che genera tale scelta;
- ottimizzazione delle componenti di un environment per l'operatività in virtualizzazione;
- hardening dell'ambiente delle virtual machine, per garantire un livello di security più elevato; ad esempio, firewalling dinamico e supporto alle VLAN, senza rinunciare all'ibernazione e migrazione;
- supporto e gestione dell'hotplug di periferiche per le virtual machine;

- approfondire gli aspetti a basso livello della virtualizzazione;
- realizzazione di un batch system per l'impiego in virtualizzazione, che gestisca il fairshare in modo intelligente;
- . . . (etc.)

La lista non aveva la pretesa di essere esaustiva, ma piuttosto di invogliare il lettore che è giunto fin qui ad impegnarsi in prima persona; se proprio non vuole, peggio per lui.

Bibliografia

[1] Ian Foster e Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, November 1998.

[2] AA.VV. glite 3.1 user guide - `https://edms.cern.ch/file/722398/1.2/gLite-3-UserGuide.html`.

[3] CERN. Scientific Linux Cern 4 (SLC4) - `http://linux.web.cern.ch/linux/scientific4/`.

[4] CERN. QUATTOR - `http://quattor.web.cern.ch/quattor/`.

[5] Platform Computing. LSF documentation - `http://batch.web.cern.ch/batch/doc-lsfsets.html`.

[6] IBM. Ibm General Parallel File System - `http://www.ibm.com/systems/clusters/software/gpfs.html`.

[7] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. In *SOSP '73: Proceedings of the fourth ACM symposium on Operating system principles*, volume 7, New York, NY, USA, October 1973. ACM Press.

[8] University of Cambridge Computer Laboratory. Xen documentation - `http://www.cl.cam.ac.uk/research/srg/netos/xen/documentation.html`.

[9] Trinity College Dublin. pypxeboot: a pxe bootloader for xen guests - `https://www.cs.tcd.ie/Stephen.Childs/pypxeboot/`.

[10] INFN Parma. INFNGrid Parma's wiki - `http://www.fis.unipr.it/dokuwiki/doku.php?id=grid:start`.