UNIVERSITA' DEGLI STUDI DI CAGLIARI

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea in Tecnologie Informatiche

Un'infrastruttura distribuita per il calcolo Grid e Cloud

Relatore:
Prof. Andrea Bosin

Candidato:
Giovanni Serra

ANNO ACCADEMICO 2010/2011

INDICE

INTRODUZIONE

Il lavoro di questa tesi focalizza l'attenzione su due diversi paradigmi di calcolo distribuito, con particolare attenzione alle diverse fasi di installazione, configurazione e gestione di un cluster di computer dedicato al Grid e Cloud computing. La fase di sperimentazione, è stata svolta utilizzando le risorse del polo di calcolo CyberSAR sito nella facoltà di Ingegneria a Cagliari. Tale infrastruttura inizialmente utilizzata per il calcolo HPC ed il Grid computing, in seguito ad esigenze di ottimizzazione dell'uso delle risorse e riduzione generale dei consumi, ha subito un processo di evoluzione e di trasformazione in cui sono state impiegate tecniche di virtualizzazione che hanno permesso di eseguire anche funzioni di Cloud computing. Alcune soluzioni adottate hanno lo scopo di limitare il più possibile l'intervento dell'amministratore di sistema, automatizzando e ottimizzando alcune operazioni e procedure, evidenziando le problematiche relative alla gestione di un polo di calcolo non presidiato ma gestito quasi totalmente in maniera remota dal polo CyberSAR di Monserrato. Particolare attenzione, verrà posta sull'analisi delle due applicazioni di Cloud computing utilizzate in modo da mettere in risalto alcuni miglioramenti apportati durante la fase di sperimentazione. Alcuni dei risultati ottenuti sono stati inseriti nell'articolo scientifico "Enhancing Eucalyptus Community Cloud", in via di pubblicazione sulla rivista scientifica online "http://www.scirp.org/journal/iim". Di seguito vengono elencate le varie fasi del lavoro di tesi, organizzato in sei parti ognuna delle quali corrisponde ad un capitolo:

Capitolo primo: si fornisce una panoramica generale del progetto CyberSAR e delle sue finalità.

Capitolo secondo: viene descritto in maniera dettagliata il polo di calcolo CyberSAR dove sono state eseguite le sperimentazioni oggetto di questa tesi e in particolare l'infrastruttura di calcolo e gli strumenti utilizzati per il suo funzionamento.

Capitolo terzo: viene fornita una panoramica sul concetto di calcolo distribuito, sull'installazione e la configurazione di un cluster di computer dedicato al Grid computing, focalizzando l'attenzione sull'utilizzo di "Sun Grig Engine" come job manager efficiente e affidabile.

Capitolo quarto: illustra le principali tecniche di virtualizzazione spiegando i motivi che nella fase di sperimentazione hanno portato a scegliere l'utilizzo di una certo strumento piuttosto che un altro.

Capitolo quinto: descrive in generale il funzionamento del Cloud computing e le funzionalità dei software adoperati per la sperimentazione di tale sistema di calcolo "OpenNebula" e "Eucalyptus", riportando i risultati dei test eseguiti e descrivendo i miglioramenti conseguiti.

Capitolo sesto: presenta le conclusioni finali mostrando un confronto sia tra il Grid ed il Cloud computing che tra le applicazioni di Cloud computing utilizzate.

Allegato A: viene riportato integralmente l'articolo "Enhancing Eucalyptus Community Cloud".

1. IL PROGETTO CYBERSAR

Il progetto CyberSAR[1], finanziato dal Ministero dell'Università e Ricerca su fondi "PON Ricerca", è nato dall'esperienza acquisita con la partecipazione a grandi collaborazioni e progetti europei nell'ambito delle cosiddette "e-infrastrutture" per la ricerca. CyberSAR è stato classificato al primo posto per la qualità della proposta come progetto di eccellenza dall'Autorità del MIUR. Il consorzio CyberSAR (Consorzio per il supercalcolo, la modellistica computazionale e la gestione di grandi database), soggetto proponente del progetto, è costituito dai seguenti soci:

- UNICA: Università degli Studi di Cagliari;
- UNISS: Università degli Studi di Sassari;
- INAF: Istituto Nazionale Astrofisica;
- INFN: Istituto Nazionale Fisica Nucleare;
- CRS4: Centro di Ricerca, Sviluppo e Studi Superiori in Sardegna;
- NICE s.r.l. (AT);
- TISCALI Italia s.r.l. (CA).

E' finalizzato alla realizzazione di una cyber-infrastructure in Sardegna, organizzata su una rete di poli di calcolo ad alte prestazioni ed inizialmente basata sulla tecnologia del Grid computing. Gli obiettivi sono orientati alla ricerca fondamentale ed applicata nei settori scientifici delle scienze naturali, dell'ingegneria e dell'informatica. L'infrastruttura è basata su poli di calcolo complementari, localizzati in modo da rafforzare e valorizzare pre-esistenti infrastrutture di ricerca ed integrati in una architettura di tipo Grid. Ogni singolo polo è concepito, dimensionato e realizzato in modo da poter risolvere problemi computazionali a grande scala e tramite connessioni in fibra ottica dedicate permettono di sperimentare nuovi paradigmi di calcolo in grado di aggregare dinamicamente risorse distribuite.

1.1 Infrastruttura fisica

L'infrastruttura fisica è basata su quattro poli di calcolo principali, interconnessi da reti ad alta velocità a cui si aggiungono una serie di poli minori dedicati ad attività specifiche nei quali si prevede di realizzare le seguenti attività:

- Il polo localizzato presso il campus universitario di Monserrato. E' usato principalmente per lo sviluppo di nuovi algoritmi, codici e metodi di simulazione HPC (High Performance Computing), per la ricerca nella fisica computazionale della materia dura e soffice, per la ricerca nella fisica nucleare, subnucleare e delle alte energie insieme all'esperimento ALICE (A Large Ion Collider Experiment del CERN di Ginevra) e LHCb (Large Hadron Collider beauty). Inoltre sono presenti dei gruppi di lavoro che si occuperanno dello sviluppo di tecniche di simulazione

modellistica per la soluzione di problemi a grande scala nella pianificazione territoriale, ambientale e dei servizi. Pertanto l'infrastruttura computazionale di questo polo sarà orientata principalmente al supporto di applicazioni computazionali di tipo "number crunching". Alcune piattaforme di calcolo di questo polo sono state allocate presso il campus universitario di Cagliari. Questo è il caso del polo di calcolo di ingegneria, situato presso la facoltà di Ingegneria Elettronica a Cagliari in Piazza d'Armi;

- Il polo localizzato presso il parco scientifico e tecnologico Polaris. E' usato principalmente per lo studio e lo sviluppo di tecniche e sistemi ad high throughput per applicazioni di simulazione, data-mining e visualizzazione "data intensive". L'infrastruttura computazionale di questo polo privilegia architetture cluster con un alto grado di parallelismo e con notevoli capacità di I/O distribuito ad alta velocità;
- Il polo localizzato presso la sezione INAF di Cagliari, Osservatorio Astronomico di Cagliari (OAC). E' usato principalmente per lo sviluppo di una serie di tecniche avanzate per l'elaborazione dati in tempo reale, attraverso la connessione dedicata ad un front-end localizzato presso il nuovo radiotelescopio (SRT) di San Basilio, per l'analisi multidimensionale di esperimenti di survey radio profonde, per la simulazioni e la modellistica di processi astrofisici e per analisi correlate tramite l'accesso ai grandi database astronomici internazionali. L'infrastruttura del polo e del suo front-end saranno pertanto articolate in varie tipologie di sottosistemi, la cui architettura è determinata in funzione delle problematiche connesse al flusso di dati e alle richieste computazionali "real-time" della rete interferometrica internazionale della radioastronomia, alle esigenze di data processing "off-line" di grandi moli di dati collezionate nelle survey radio e ottiche presso telescopi e radiotelescopi della rete internazionale;
- Il polo localizzato presso il campus universitario di Sassari. Rappresenta il punto computazionale di presenza del consorzio nel nord della Sardegna, finalizzato principalmente allo sviluppo di nuovi algoritmi, codici e metodi di simulazione HPC per la ricerca nella chimica computazionale dei sistemi complessi, orientata allo studio di materiali sia di interesse chimico che di interesse agro-biologico (componenti del terreno, membrane) nonché allo sviluppo di tecniche di archiviazione e estrazione automatica di informazioni clinicamente rilevanti da database di immagini mediche e all'analisi di sequenze multimediali orientata all'identificazione, riconoscimento ed autenticazione automatica di persone.

1.2 Infrastruttura di rete

L'infrastruttura di rete è pensata per poter garantire sia altissime velocità di connessione tra i poli utilizzando tecnologie standard che per permettere attività di ricerca e sperimentazione su lambda-Grid e Bandwidth Unlimited Computing. A tal proposito, nell'ambito del progetto, è stato realizzato l'accesso ad alta velocità alla rete telematica regionale per la ricerca. Si tratta di una griglia computazionale di rilevanza europea servita da un'infrastruttura di rete ad alta velocità capace di interconnettere, il centro stella situato a "Sa Illetta" in cui è presente il nodo di terminazione delle fibre ottiche sottomarine del Consorzio Janna, con la sede del CRS4 presso il Parco tecnologico della Sardegna a Pula, la facoltà di ingegneria dell'Università di Cagliari in Piazza d'Armi, la cittadella universitaria di Monserrato, la sede dell'Osservatorio Astronomico di Cagliari, (attualmente localizzato a Capoterra e di prossimo trasferimento presso la nuova sede di

Selargius in corso di realizzazione) e il radiotelescopio di San Basilio (Sardinia Radio Telescope - SRT), attualmente in costruzione. L'appartenenza alla rete telematica della ricerca, rende possibile l'integrazione dei poli regionali della ricerca e la realizzazione del nodo regionale della rete GARR. Ciò è stato possibile grazie all'acquisizione, installazione e configurazione di apparati DWDM, nello specifico uno switch ottico Alcatel 1800 TSS, capace di raggiungere velocità pari a multipli di 10Gbit/s in base alla quantità di fibre ottiche aggregate, vedi fig. 1.

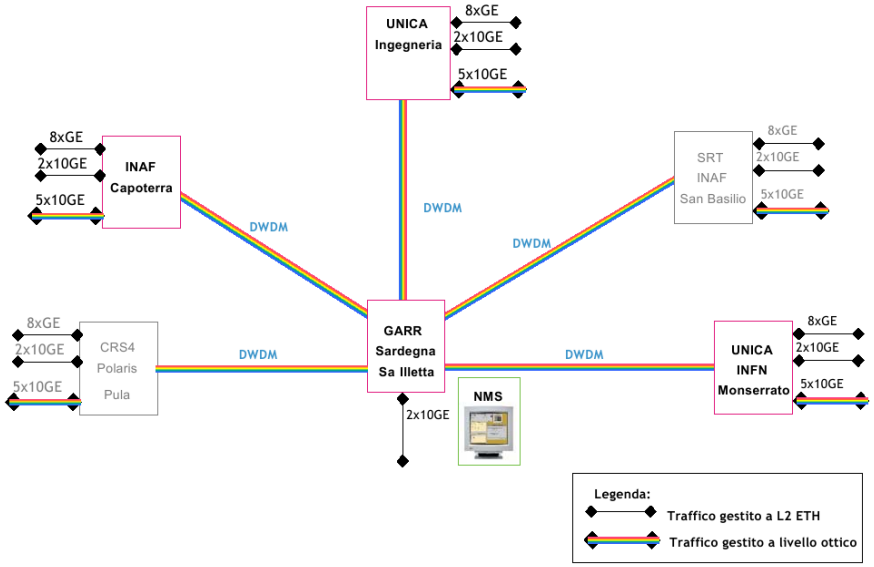

Figura 1 Connessione CyberSAR alla Rete Telematica Regionale per la Ricerca

Attraverso le fibre ottiche del consorzio Janna e Mazara del Vallo sarà possibile connettere la regione del Mediterraneo con le principali città europee ed americane attraverso MedNautilus, un broadband network services provider che gestisce una rete in fibra ottica sottomarina anch'essa in tecnologia DWDM, vedi fig. 2.

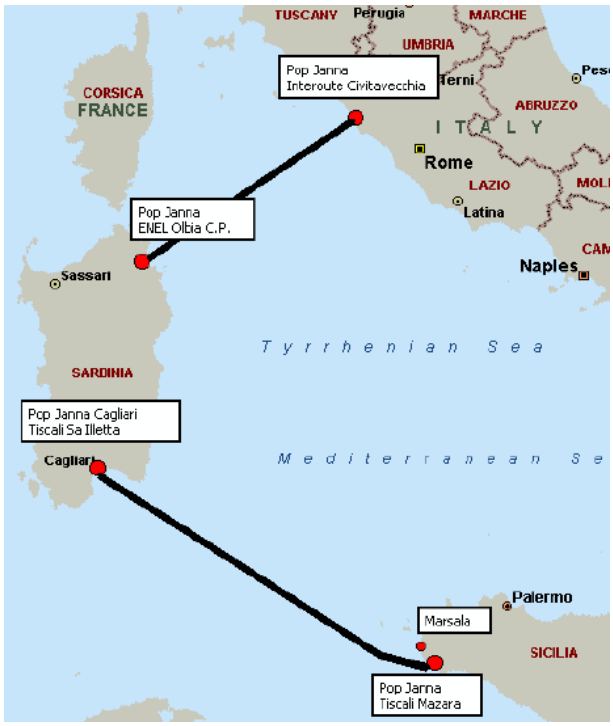

Figura 2 Fibre ottiche del consorzio Janna

Inoltre, tutti i centri coinvolti nel progetto sono connessi alla rete GARR[2], vedi fig. 3. Nell'ambito del progetto CyberSAR è stata realizzata una infrastruttura di calcolo integrata su una grid connessa ad alta velocità per la ricerca in Sardegna. Nel dettaglio, un anello a 2.5Gbit/s permette un alta connettività tra tutti i poli mentre la connessione in fibra ottica dedicata permette di realizzare un nucleo centrale ottico passivo, basato su fibre spente e permutatori ottici, in grado di permettere l'istanziazione di cammini fotonici indipendenti tra le risorse computazionali residenti nei poli. I poli meridionali sono connessi al centro stella localizzato in Sa Illetta (CA) da fibre ottiche dedicate. Questa infrastruttura fornisce risorse di calcolo che consentono alla comunità della ricerca di posizionarsi ai massimi livelli internazionali essendo integrata nell'infrastruttura di Grid nazionale ed internazionale attraverso la rete GRID-IT dell'INFN ed EGEE.

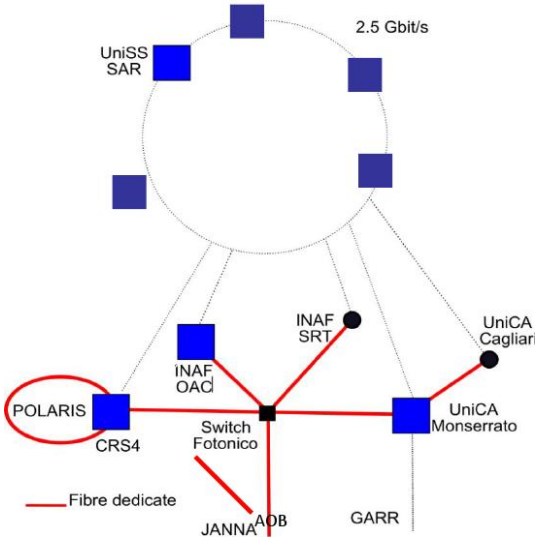

Figura 3 Connessione CyberSAR alla rete GARR

1.3 Campi di applicazione del progetto CyberSAR

- Chimica: Mdynamix è un codice parallelo scalabile di simulazione di dinamiche molecolari di sistemi molecolari di varia natura;
- Meteorologia: MOLOCH viene utilizzato per scopi di ricerca per la simulazione, ad alta risoluzione spaziale e a breve scadenza temporale, dello sviluppo di temporali, di flussi sopra orografie complesse e di processi fisici responsabili di precipitazioni intense;
- Idrologia: CODESA-3D è un modello matematico-numerico che descrive il flusso di acqua sotterranea e il trasporto di inquinanti in essa disciolti nel suolo e sottosuolo;
- Astrofisica: codice per simulazioni TD-DFT di molecole in tempo e spazio reale;
- Ingegneria strutturale ed industriale: simulazione e modellistica nell'ingegneria di processo e della produzione e la pianificazione territoriale ed ambientale;
- Fisica delle alte energie: Esperimento ALICE (A Large Ion Collider Experiment) del CERN di Ginevra;
- Fisica della Materia: CMPTool (Caspur Model Potentials Tool), applicazione multi piattaforma sviluppata per simulazioni di dinamica molecolare;

- Biofisica: ORAC è un codice di dinamica molecolare classica per simulare sistemi e processi con applicazioni nel campo biologico/medico alla scala microscopica.

2 IL POLO CYBERSAR DI INGEGNERIA

Il polo di calcolo, situato presso la facoltà di ingegneria a Cagliari in Piazza d'Armi, è il raggiungimento dell'obiettivo realizzativo A1.2 del progetto CyberSAR: "acquisizione, configurazione ed installazione di un polo computazionale presso il campus universitario di Cagliari". La data approssimativa di installazione del cluster di computer nel sito e l'inizio della sua successiva configurazione, risale a giugno/luglio del 2008. La fase di sperimentazione oggetto di questo lavoro è stata interamente eseguita utilizzando le risorse di calcolo del polo CyberSAR di ingegneria che è attualmente connesso alla rete ed operativo e gestito in maniera remota presso la facoltà di fisica di Monserrato. Le risorse del cluster inizialmente e principalmente destinate al calcolo HPC ed al Grid computing, come verrà descritto nel capitolo 3, sono in seguito state utilizzate anche per offrire servizi di Cloud computing a seguito di un processo di evoluzione e di trasformazione motivato da esigenze di ottimizzazione dell'uso delle risorse e riduzione generale dei consumi. Per il consolidamento dei servizi, è stata utilizzata la virtualizzazione delle risorse fisiche disponibili, studiando le diverse tecniche esistenti, illustrate nel capitolo 4, in modo da poter scegliere quella più adatta alle finalità da realizzare. Ciò ha permesso di sfruttare le funzionalità offerte dalle applicazioni "OpenNebula" e "Eucalyptus", descritte nel capitolo 5, utilizzate sia come front-end per i servizi di Cloud che dalla Digital Library del progetto E-Pistemetec[3] di cui il consorzio CyberSAR fa parte. Le scelte sulla dotazione hardware, i dispositivi di rete, i sistemi operativi ed i vari software e toolkit utilizzati, sono state effettuate e modificate nel corso del tempo in modo da poter soddisfare tali requisiti. Viene di seguito mostrato un riepilogo generale della configurazione hardware presente nel cluster, vedi fig. 1:

- n.1 armadio rack IBM 42U, completo di unità di alimentazione (PDU);
- n.1 nodo di calcolo IBM System x3650 completo di 2 processori Intel Xeon Quad Core E5420, RAM 2 GB DDR2 PC2-5300 667 MHz, 2 dischi HS SATA 500 GB 7200 RPM, schede di rete ethernet 10/100/1000 PCI-E 4x Broadcom NetXtreme II, scheda IPMI, unità DVD/CD-RW, alimentazione ridondata;
- n.1 nodo di calcolo Supermicro Superserver 5015B-MTB completo di 2 processori Inter Core 2Duo E8400 3 Ghz/1333MHz/6Mb L2, RAM 4 GB DDR2 PC2-5300 667 Mhz, 4 dischi HS SATA 2 TB 7200 RPM, scheda IPMI;
- n.20 nodi di calcolo IBM System x3550 completi di 2 processori Intel Xeon Quad Core E5420, RAM 8 GB DDR2 PC2-5300 667 MHz, 2 dischi HS SATA 250 GB 7200 RPM, schede di rete ethernet 10/100/1000 PCI-E 4x Broadcom NetXtreme II, schede InfiniBand SilverStorm 9000 DDR HCA, scheda IPMI, unità DVD/CD-RW;
- n.1 unità KVM 16 porte IBM GCM2;
- n.1 console KVM per armadio rack competa di monitor LCD 17" e tastiera;
- n.1 switch Gigabit ethernet 48 porte Cisco 4948-10GE completo di 2 moduli Cisco

10GBASE-SR X2;

- n.1 switch ethernet 24 porte Cisco 2960G;
- n.1 switch InfiniBand 24 porte SilverStorm 9024 DDR;
- n.1 terminal server 48 porte Cyclades.

Figura 1 Cluster del polo CyberSAR di ingegneria

2.1 Architettura hardware

I componenti fondamentali di un cluster sono quelli relativi al calcolo, alla gestione, alla memorizzazione dei dati persistenti e alla rete d'interconnessione tra nodi interni e tra il cluster e l'esterno. Ormai, i nodi di calcolo vengono equipaggiati dai maggiori produttori di hardware con processori multi-core. Ciò permette di ottenere prestazioni e scalabilità di esecuzione del codice sempre maggiori, a condizione che nel nodo sia installata una quantità di memoria RAM adeguata al numero di core presenti. In generale, avere più core per nodo, significa dover avere più memoria RAM per nodo. E' stato necessario destinare una macchina alla funzione di server, nello specifico l'IBM System x3650 date le sue caratteristiche. Questo nodo, quindi, non eseguirà

calcoli sui jobs sottomessi dagli utenti, ma esporrà servizi sia verso l'esterno del cluster che al suo interno. Dato il numero ridotto di nodi a disposizione, si è scelto di concentrare quanti più servizi possibile in questa macchina in modo da riservare gli altri nodi al calcolo. Il server inoltre, dovrà disporre di almeno una interfaccia di rete con indirizzamento pubblico per le richieste provenienti dall'esterno, ed almeno una interfaccia con indirizzamento privato per le richieste provenienti dai nodi interni. Oltre alle caratteristiche hardware, gli altri 20 nodi si distinguono in base alle funzionalità svolte all'interno del cluster, poiché eseguiranno direttamente i jobs sottomessi dagli utenti. Qualsiasi cluster, ed in particolar modo se dedicato al calcolo distribuito come quello in oggetto, ha bisogno di un qualche tipo di storage ad alte prestazioni. Nello specifico le applicazioni HPC solitamente creano una grossa mole di dati, il Cloud computing necessita almeno di un repository per la memorizzazione delle immagini ed anche la Grid, pur non avendone configurato alcuno ad ingegneria, ha bisogno di un componente dedicato allo storage. Per cui è di fondamentale importanza avere un sistema di memorizzazione affidabile, integrato e compatibile con l'hardware del resto del cluster. Uno dei metodi più semplici è quello di utilizzare un nodo come server NFS e configurare i dischi in una delle varianti RAID. Questa soluzione presenta però problemi di scalabilità con il crescere del numero dei nodi. E' stato necessario quindi dedicare il nodo Supermicro 5015B-MTB come server per lo storage, adottando una soluzione basata su di un filesystem parallelo distribuito più efficiente e robusto come GPFS. La dotazione di schede IPMI (Intelligent Platform Management Interface) in tutti i nodi, oltre al monitoraggio di numerosi parametri quali frequenza di funzionamento delle CPU e temperature, ha consentito la gestione di tutta l'infrastruttura da remoto comprese le operazioni di accensione, spegnimento e riavvio. In questo modo, guasti hardware a parte, è stato possibile quasi sempre evitare di recarsi fisicamente nel sito che ospita il cluster. Di seguito vengono illustrati i compiti principali assegnati ad ogni nodo del cluster.

Il server principale svolge diverse ed importanti funzioni quali:

- Server LDAP (Lightweight Directory Access Protocol): viene usato per una gestione centralizzata e uniforme degli utenti in una rete eterogenea. Tale scelta è giustificata dall'esigenza di catalogare in un unico punto tutte le informazioni riguardanti gli oggetti presenti in un ambiente distribuito, ad esempio gli utenti, garantendo il controllo sugli accessi tramite un processo di autenticazione;
- Server NTP (Network Time Protocol): viene usato per sincronizzare l'orologio interno di tutti i nodi del cluster. Durante la comunicazione tra nodi, si potrebbero verificare grossi problemi se non ci fosse corrispondenza sul timing delle diverse operazioni eseguite. E' quindi di fondamentale importanza che il riferimento temporale sia unico e sempre consistente;
- Server VPN (Virtual Private Network): viene usato per il monitoraggio in maniera remota di alcuni parametri del cluster visualizzabili più comodamente tramite interfaccia web. L'instaurazione di una rete privata tra server e client VPN consente che tale traffico possa essere visualizzato da remoto attraverso una connessione sicura di tipo punto-punto o tunnel;
- Gateway: è il punto di uscita verso l'esterno per la rete interna privata dei nodi, per il traffico di management e per la rete dedicata all'IPMI;
- UI (User Interface): è il punto di accesso al cluster per gli utenti Grid. Dopo aver fatto l'accesso alla UI, l'utente può essere autenticato e autorizzato all'uso delle risorse messe a disposizione dalla Grid. Tramite interfaccia a riga di comando, sarà

possibile eseguire diverse operazioni come listare tutte le risorse necessarie per l'esecuzione di un job, sottomettere e cancellare un job, mostrarne lo stato o recuperarne l'output;

Il server di storage svolge le seguenti funzioni:

- Server GPFS: gestisce i dischi e le relative partizioni che verranno montate dai nodi del cluster in modo da consentire alle applicazioni che sfruttano il calcolo parallelo l'accesso simultaneo ai file, offrendo un elevato livello di controllo su tutte le operazioni del filesystem;
- Backup delle "home" degli utenti: vengono memorizzati i dati degli utenti per il recupero di una eventuale perdita accidentali dei dati;
- Backup delle immagini del sistema operativo: vengono memorizzate le immagini del sistema operativo usate nel cluster. In questo modo, in caso di gravi problemi di funzionamento dovuti a problemi hardware o a configurazioni errate, si è in grado in qualsiasi momento di ripristinare velocemente una configurazione precedentemente funzionante;
- Repository per il sistema di Cloud: vengono memorizzate le immagini usate come istanza per le macchine virtuali invocate dagli utenti.

I restanti nodi, chiamati anche worker nodes (WN), sono utilizzati per soddisfare le esigenze di calcolo degli utenti, che in generale ricadono in due tipologie di esecuzione: i jobs seriali e quelli paralleli. Le elaborazioni eseguite riguardano principalmente sottomissioni provenienti da utenti Grid nell'ambito dell'esperimento ALICE e LHCb. Il controllo delle risorse disponibili e delle priorità di esecuzione, sono gestite da due job manager: Sun Grid Engine (SGE) e Load Sharing Facility (LSF). Ovviamente si è fatto in modo che ognuno di essi abbia delle specifiche code di esecuzione e possa controllare nodi diversi, in modo da gestire le risorse in maniera esclusiva. In questo modo all'interno di un unico cluster metà dei WN sono gestiti da SGE e metà da LSF. Tale scelta consente all'occorrenza di poter unire le risorse del polo di ingegneria a quelle del polo di Monserrato e come verrà descritto meglio più avanti, evitare la replica di servizi già esistenti con il conseguente uso più efficiente delle risorse disponibili. Per rendere indipendente dal punto di vista della Grid il polo di ingegneria, si è dovuto destinare e configurare uno dei nodi per svolgere le seguenti funzioni:

- CE (Computing Element): in ambito Grid è l'insieme delle risorse di calcolo localizzate nel cluster o sito a cui il CE fa riferimento;
- SGE Master: il job manager SGE necessita di un nodo master che gestisca tutte le richieste di sottomissione dei vari jobs sottomessi dagli utenti;
  Shadow Server LDAP: viene usato come server LDAP secondario nel caso in cui non sia disponibile quello principale.

2.2 Architettura di rete

Le tecnologie di rete oggi disponibili per realizzare l'interconnessione dei nodi, sono molte e varie. La scelta delle componenti di rete, dipende da vari fattori molto importanti quali: i tempi di latenza, l'ampiezza di banda, la compatibilità tra i diversi dispositivi del cluster e i costi. Spesso la tipologia di rete utilizzata è basata sullo standard de facto Ethernet che rappresenta un compromesso nell'ottimizzazione di tali

fattori. In particolare la versione Gigabit, è in grado di offrire ampiezza di banda di 1Gbps, con costi relativamente ridotti e tempi di latenza minimi dovuti solo all'overhead dei protocolli di livello superiore. Esistono, però, anche tecnologie più costose che possono offrire vantaggi in termini di prestazioni nella comunicazione tra i nodi come il 10 Gigabit Ethernet, FibreChannel, le reti Myrinet della Myricom oppure soluzioni InfiniBand capaci di throughput ancora maggiori. La topologia della rete dipende fortemente dalla tecnologia di rete utilizzata. E' possibile comunque utilizzare più tecnologie contemporaneamente su diverse porzioni dell'infrastruttura in modo da soddisfare requisiti particolari e poter differenziare i diversi tipi di traffico. E' molto importante diversificare le diverse tipologie di traffico esistenti all'interno del cluster. Per raggiungere tale risultato, sono stati utilizzati diversi dispositivi di rete e si sono dovute configurare alcune VLAN. Nel cluster di ingegneria i flussi di rete sono stati distinti in questo modo: traffico per la gestione IPMI, traffico di management e monitoraggio, traffico per la comunicazione interna ai nodi, traffico dati e traffico di storage, traffico dati a bassa latenza. Di seguito vengono descritti i dispositivi di rete installati nel cluster e le relative funzioni da loro svolte, vedi fig. 2:

- switch Cisco Catalyst 3750-24TS: è uno switch a 24 porte con funzionalità di livello 3. Date le sue caratteristiche viene utilizzato anche come router per instradare il traffico da e verso la connessione in fibra ottica per la rete GARR;
- switch Cisco Catalyst 4948-10GE: è uno switch a 48 porte più ulteriori due porte in fibra con tecnologia 10 Gigabit Ethernet. Grazie alle porte a 10 Gbps, viene utilizzato per gestire il traffico di storage e la rete privata interna ai nodi;
- switch Cisco Catalyst 2960G: è uno switch in tecnologia Gigabit Ethernet a 24 porte, di cui 4 con connettore in fibra ottica. Gestisce il traffico di management e monitoraggio;
- switch 3Com SuperStack 3300: è uno switch in tecnologia Fast Ethernet a 24 porte. Poiché non è dotato di una altissima velocità è stato usato per la gestione del traffico IPMI;
- switch InfiniBand SilverStorm 9024: è uno switch in tecnologia Infiniband a 24 porte che supporta la tecnologia DDR (Double Data Rate). InfiniBand è una tecnologia particolarmente adatta all'uso in ambiente HPC, viene infatti, utilizzata per la gestione del traffico a bassissima latenza proveniente da tutti i WN ad esso collegati, permettendo di raggiungere una larghezza di banda pari a 20 Gbps;
- switch Alcatel 1830: switch fotonico in tecnologia fibra ottica, utilizzato per la gestione del traffico della rete telematica regionale (RTR). Rende possibile l'integrazione dei poli regionali della ricerca, in particolare connette il polo CyberSAR di ingegneria con quello di Monserrato con una larghezza di banda pari a 10 Gbps.

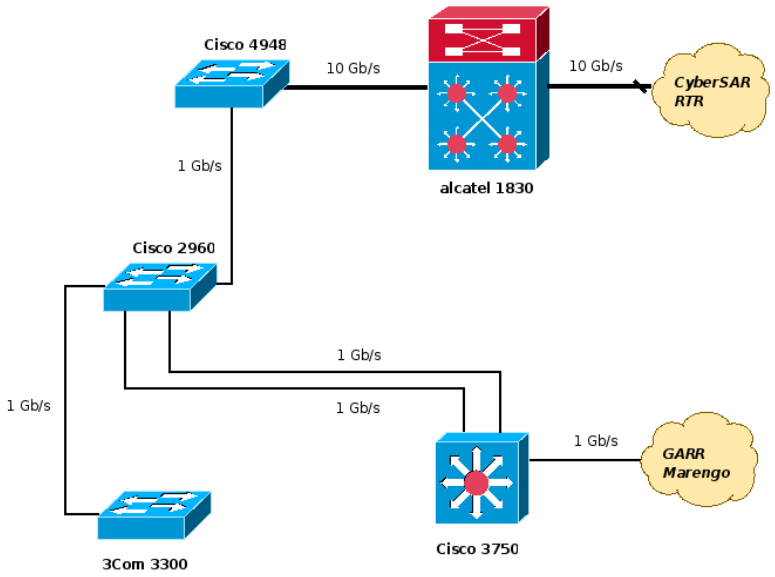

Figura 2 Architettura di rete del polo CyberSAR di ingegneria

2.3 Caratteristiche Software

Ogni nodo, essendo un sistema autonomo, ha bisogno di eseguire un'istanza indipendente del sistema operativo. Ciò implica che i singoli nodi non hanno una diretta conoscenza delle risorse disponibili nell'intero cluster. E' però necessario che l'intera struttura di calcolo possa essere percepita come un unico sistema dagli utenti senza la necessità che questi debbano conoscere i dettagli dell'architettura del sistema stesso. Vengono, quindi, impiegati dei meccanismi, utilizzati come supporto per sistemi distribuiti complessi, che permettono l'astrazione dalla granularità dovuta all'impiego di più sistemi di elaborazione indipendenti, vedi fig. 3. Nel caso del Grid computing, questa astrazione è realizzata da una o più componenti, che costituiscono un'ulteriore strato software detto middleware. Nel capitolo 3, verrà descritto in maniera dettagliata il funzionamento di gLite, il middleware scelto per il calcolo eseguito tramite Grid computing.

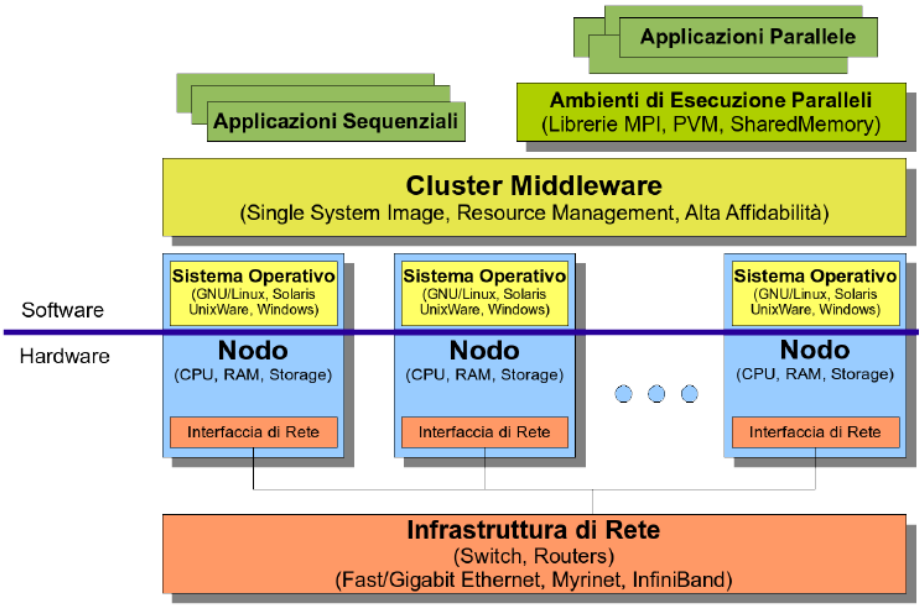

Figura 3 Schema ad alto livello di astrazione dell'architettura hardware e software di un cluster di computer

2.3.1 Scelta del sistema operativo Linux

Abbiamo già accennato al fatto che ogni singolo nodo esegue una istanza indipendente del sistema operativo per la gestione a basso livello delle risorse che condivide con il resto del cluster. Il sistema operativo adottato, deve supportare i comuni protocolli di comunicazione e gli standard nei paradigmi di programmazione (multitasking e multithreading, RPC, sockets, ecc.), in modo da garantire una piena interoperabilità in ambienti distribuiti come quello dei cluster. Per questo motivo, molte implementazioni di cluster, utilizzano sistemi operativi a sorgente aperto basati sullo standard POSIX, come ad esempio GNU/Linux e xBSD, con opportune modifiche per adattarli ad un ambiente distribuito e parallelo. In particolare, anche la scelta del middleware ha avuto importanti conseguenze sulla scelta del sistema operativo e la versione di kernel da utilizzare. Attualmente, sono installate le distribuzioni Scientific linux 4.6 (con kernel 2.6.9) e Scientific linux 5.3 (con kernel 2.6.18). Tali scelte sono state fatte sopratutto per rispettare e soddisfare i requisiti imposti dalla versione del middleware di Grid correntemente installata e cioè gLite 3.2. Le principali motivazioni che hanno portato a scegliere come sistema operativo una distribuzione Linux, oltre a quelle descritte in precedenza sono le seguenti:

- Vasta eterogeneità: copertura di una vasta gamma di architetture, piattaforme e dispositivi;
- Ampia diffusione: la maggior parte delle attuali soluzioni cluster sono realizzate con sistemi Linux;
- Codice sorgente aperto: il sistema operativo Linux è open-source, caratteristica molto importante sia per avere una piena visione del codice che corrisponde ad una piena libertà di modifica, sia per il supporto garantito da una vasta comunità di utenti e sviluppatori;
- Compatibilità verso altri sistemi: possibilità di gestire anche sistemi non Linux.

Alcuni sistemi, infatti, hanno una piena compatibilità solo con ambienti dello stesso tipo, mentre uno dei maggiori sforzi fatti da Linux è stato quello di massimizzare la compatibilità con altri sistemi operativi (UNIX e non UNIX).

2.4 Problematiche di gestione relative ad un sito non presidiato

L'amministrazione di un cluster, anche se di dimensioni ridotte come quello in oggetto, ha richiesto l'utilizzo di diversi strumenti di gestione in modo da semplificare ed automatizzare la maggior parte delle operazioni e permettere di replicare una stessa configurazione o funzionalità su macchine diverse. Tali necessità diventano ancora più stringenti in un sito non presidiato, in quanto la sede in cui è collocato il cluster è distante circa 15 km dagli uffici da cui viene amministrato in modalità remota. Per le stesse motivazioni risulta fondamentale disporre di un sistema di monitoraggio efficiente e affidabile che sia in grado di spegnere in maniera parziale o totale i nodi del cluster in caso di necessità. I problemi più critici riguardano il raggiungimento di temperature troppo alte e la mancanza di energia elettrica per un periodo di tempo prolungato. Per ovviare a questi due inconvenienti le temperature vengono costantemente monitorate tramite alcuni sensori collegati ad una centralina di controllo. I sensori di rilevamento della temperatura, forniscono i valori di quest'ultima in ingresso ed in uscita ai nodi del cluster mentre un sensore di rilevamento di segnale, collegato ad una scheda di comunicazione presente nell'UPS, intercetta la mancanza di energia elettrica. Questi dati, memorizzati nel server principale, vengono elaborati da una serie di script eseguiti periodicamente e in caso di superamento dei valori limite, fanno partire automaticamente uno spegnimento controllato dei dispositivi in quel momento in funzione, seguendo una determinata sequenza.

2.4.1 SystemImager

L'operazione di installazione o aggiornamento di un sistema operativo in un cluster e l'esecuzione dei backup, può richiedere molto tempo e essere soggetta ad errori in quanto frequente e ripetitiva. Automatizzare queste procedure, significa aumentare la produttività inerente le attività in corso e ridurre i tempi di manutenzione in cui alcuni servizi, principali o secondari diventano non disponibili. SystemImager[4] è un progetto open-source finalizzato all'automazione di installazione e aggiornamento di sistemi operativi GNU/Linux su ambienti distribuiti complessi. L'approccio utilizzato è quello client-server, in cui le immagini vengono recuperate da una macchina appositamente configurata, chiamata "golden client" e memorizzate su di un server centralizzato. Queste immagini, rappresentano lo stato del filesystem da clonare esattamente così come dovranno comparire sulla macchina di destinazione, includendo tutti i file e le directory partendo dalla radice. Ad installazione terminata, le immagini vengono depositate in automatico sui client di destinazione. Oltre al vantaggio di poter effettuare le installazioni dei sistemi operativi da remoto in maniera automatica, SystemImager è un ottimo strumento di backup, poiché permette di ripristinare in maniera relativamente facile e veloce una configurazione precedente funzionante. Una limitazione, di cui tener conto e che incide in termini di prestazioni, è che il tempo totale di installazione dipende linearmente dal numero di client installati contemporaneamente. L'"image server" è l'host in cui verranno memorizzate le

immagini provenienti dai client. Tale funzione è svolta dal server principale del cluster, poiché già configurato come DHCP e TFTP server, indispensabili per il corretto funzionamento di SystemImager. Prima di procedere con qualsiasi operazione è necessario editare il file principale di configurazione solitamente posizionato in "/etc/systemimager/systemimager.conf", in cui devono essere indicate importanti informazioni come la posizione delle directory in cui verranno memorizzate le immagini di default ed dei file relativi al TFTP per il boot via rete. Di seguito viene descritto un esempio di utilizzo nel cluster in cui per "golden client" si intende l'host di cui viene creata l'immagine:

- Installazione di "systemimager-server" nell'"image server" e di "systemimager-client" nel "golden client": sono disponibili nel sito "http://wiki.systemimager.org" in versione pacchettizzata come "deb" o "rpm", oppure è possibile scaricare i sorgenti per una successiva compilazione;
- Creazione dell'immagine del "golden client" e memorizzazione nel "image server": nel "golden client" eseguire il comando:
  # prepareclient --server <image-server ip address>
  in modo da indicare l'host da cui verrà fatta l'immagine. Il processo di trasferimento dei file nel "image server" può essere eseguito con il comando:
  # getimage -golden-client <golden-client ip address> -image <image-name> -ip-assignment <parameter>
  dove uno dei parametri può essere:
  - static_dhcp: un server DHCP assegnerà ogni volta lo stesso indirizzo IP statico ed il corrispondente hostname ai clients installati con questa immagine. Per assegnare valori diversi basterà modificare il file di configurazione del DHCP "dhcpd.conf";
  - dynamic_dhcp: un server DHCP assegnerà gli indirizzi IP dinamicamente ai clients installati con questa immagine;
  - static: l'indirizzo IP usato dal client durante l'installazione sarà assegnato in maniera permanente al client stesso;
  - replicant: non viene assegnato nessun indirizzo IP all'immagine.

  Una volta creata, l'immagine, questa potrà essere installata in tutti i nodi del cluster. Uno dei limiti più grandi è quello di non permettere la creazione di immagini di filesystem se non attraverso la clonazione da un golden client.
- Installazione di una immagine precedentemente clonata su un client: configurare il server DHCP in modo che sia in grado di fornire l'indirizzo IP in base all'indirizzo MAC dell'host da installare. Configurare il server TFTP in modo che fornisca i dati per l'installazione, tramite protocollo PXE (Preboot Execution Environment), dell'immagine del sistema operativo clonata in precedenza che si intende installare. Nel "image server" eseguire il comando:
  # /etc/init.d/systemimager-server start
  oppure:
  # . /etc/init.d/systemimager-server-rsyncd start
  A questo punto è possibile eseguire il boot dell'host in cui effettuare l'installazione, poiché sarà compito di SystemImager fornire i corretti file di avvio e installazione;
- Cancellazione di una immagine: nel "image server" eseguire il comando:
  # rmimage <image_name>

2.4.2 LDAP

La gestione di numerosi utenti su servizi informatici eterogenei e distribuiti richiede la creazione di innumerevoli account su macchine diverse e per scopi diversi. Questo rende molto difficile la manutenzione degli account, con problemi di sicurezza generale del sistema informatico: account validi non usati, account non più validi, password da cambiare su più macchine, ecc. Centralizzare la gestione delle utenze, risolve il problema permettendo di mantenere coerente e consistente su più macchine lo stato degli account operando da una singola postazione. LDAP[5] (Lightweight Directory Access Protocol) è un protocollo client-server per l'accesso ai servizi di directory, in particolare ai servizi basati sullo standard internazionale X-500. LDAP funziona sopra il protocollo TCP/IP o su altri protocolli di rete orientati alla connessione. Le informazioni gestite da questo protocollo, sono organizzate gerarchicamente ad albero in base ad una serie di attributi. Nel server principale è stata installata la versione open-source "OpenLDAP", utilizzata per la gestione centralizzata degli utenti. Dopo aver eseguito l'installazione dei pacchetti specifici per la parte server e quella client, si può proseguire con la configurazione. Il tool grafico "system-config-authentication" permette di configurare velocemente i parametri per la connessione al server LDAP. Il meccanismo di autenticazione usato da LDAP è PAM (Pluggable Authentication Module), una tecnologia che rende trasparente il meccanismo di autenticazione alle applicazioni che necessitano di autenticare gli utenti. I server LDAP, possono essere replicati per meglio garantire la funzionalità del servizio. Il principale file di configurazione lato server è "/etc/openldap/slapd.conf" e consiste in una serie di opzioni di configurazione globali che vengono applicate interamente a "slapd", il demone principale del server LDAP. Un altro processo in esecuzione nel server è "slurpd", responsabile del mantenimento e della sincronizzazione della replica del server stesso. Le directory "/etc/openldap/schema" e "/var/lib/ldap", contengono rispettivamente le strutture dei dati e i databases utilizzati. Le voci registrate nel database vengono inserite mediante l'impiego di appositi file ASCII in formato testo, denominati LDIF (LDAP Data Interchange Format), all'interno dei quali vengono definite le "entries" e i loro attributi. Una "entry" è individuata in maniera univoca da un Distinguished Name (DN) e affinché le venga attribuito il significato di account, deve rispettare un particolare formato o schema, definito dall'attributo "objectclass" nel file "/etc/openldap/schema/". I principali comandi che consentono lo scambio di informazioni tra server e client sono:

- ldapadd: apre una connessione con il server LDAP, collega, modifica e aggiunge le entries;
- ldapsearch: apre una connessione con il server LDAP, collega ed esegue una ricerca usando parametri specifici;
- slapadd: aggiunge voci specificate usando il formato ".ldif" su un database LDAP;
- slapcat: apre il database LDAP e scrive le corrispondenti voci nel formato LDIF.

Di seguito viene mostrato un esempio di creazione ed inserimento di un account Unix su database LDAP tramite file ".ldif":

```
dn: uid=gserra,ou=People,dc=cyb.unica,dc=it                    primary key
uid: gserra
cn: gserra

objectClass: account                                           objectclasses
objectClass: posixAccount
```

```
objectClass: top
objectClass: shadowAccount                                    passwd info
userPassword: {SSHA}Y7ErLQTC+Xd5kc482WD1lzlvoCXXGxhd
shadowLastChange: 14287
shadowMax: 99999
shadowWarning: 7
loginShell: /bin/bash                                         shadow info
uidNumber: 506
gidNumber: 100
homeDirectory: /u/gserra
```

Per generare la password utente crittografata si può utilizzare il comando: **slappasswd**. L'inserimento della "entry" nel database è effettuato dal comando:

```
# ldapadd -x -D "cn=admin,dc=cyb.unica,dc=it" -H ldap://<server_IP_address> -W
-f file.ldif
```

Affinché l'utente inserito possa ritrovarsi sulla propria "home" directory dopo il processo di autenticazione, bisognare creare l'utente in questione sul server LDAP:

```
# mkdir /u/user_name
# cp /etc/skel/.b* /u/user_name/
# chown -R user_name.group_name /u/user_name/
```

L'eliminazione di un utente risulta abbastanza semplice:

```
# ldapdelete -vx 'uid=user_name,ou=Utenti,dc=example,dc=com' -D
"cn=admin,dc=example,dc=com" –W
```

E' possibile effettuare ricerche nel database con il comando:

```
# ldapsearch -x -D "cn=admin,dc=cyb.unica,dc=it" -b "dc=cyb.unica,dc=it"
'cn=user_name' -W
```

oppure visualizzare tutto il database LDAP:

```
# ldapsearch -x -D "cn=admin,dc=cyb.unica,dc=it" -b "dc=cyb.unica,dc=it" -W
```

2.4.3 Ganglia

La possibilità di poter monitorare diverse metriche su un numero elevato di hosts permette di avere sotto controllo lo stato generale delle risorse e una visione globale ed aggiornata dell'infrastruttura in tempo reale. Ganglia[6] è un sistema distribuito di monitoraggio per sistemi di calcolo altamente scalabile. E' stato installato nel server principale e utilizzato per controllare diversi parametri quali: carico del processore, uso della memoria, I/O e lo spazio disco. E' composto da un processo server, che risiede in una sola macchina e tanti client che comunicano al server i valori di ciascuna metrica monitorata. Il formato XML è utilizzato per la rappresentazione dei dati, XDR per la loro portabilità in maniera compatta e RRDtool per la memorizzazione e visualizzazione. Utilizza, inoltre, strutture dati e algoritmi progettati in maniera robusta per ottenere overheads per nodo molto bassi, favorire la concorrenza elevata e poter essere in grado di scalare agevolmente per gestire cluster con migliaia di nodi. Il server Ganglia, per poter funzionare necessita di un server web ed è costituito da:

- gmond: demone multi-threaded di monitoraggio, è attivo su ciascun nodo del cluster e monitora i cambiamenti nello stato dei nodi. Questo demone, usa un

semplice protocollo multicast di tipo request/response che tramite XDR raccoglie lo stato del monitoraggio e condivide queste informazioni in XML su protocollo TCP. Sostanzialmente, compie le seguenti funzionalità: monitora i cambiamenti di stato negli host e lo stato di tutti gli altri nodi ganglia attraverso un canale unicast o multicast e risponde alla richiesta con una descrizione XML dello stato del cluster;

- gmetad: è un meta demone che raccoglie i dati provenienti dagli altri "gmetad" e "gmond", fornisce inoltre, un semplice meccanismo di query per raccogliere specifiche informazioni sui gruppi di macchine. "gmetad" memorizza il loro stato in maniera compatta, grazie agli RRDtool, in un database Round Robin su disco ed esporta il sommario delle informazioni tramite XML in modo che il front-end possa visualizzarle su una pagina web. Il comportamento di "gmetad" è controllato da un singolo file di configurazione: "/etc/gmetad.conf" che deve essere eseguito sulla macchina server;
- gmetric: tool di metrica, usato per inserire metriche personalizzate applicate agli hosts monitorati da ganglia;
- gstat: tool di statistica, usato per recuperare direttamente da "gmond" le informazioni sulla metrica;
- ganglia web front-end: visualizza graficamente i dati in tempo reale raccolti da "gmetad", necessita di un server web poiché è basato su una interfaccia web scritta in linguaggio PHP. Fornisce fino a tre livelli di visualizzazione dei dati: uno per la grid: multi-cluster view, uno per ogni cluster: physical view e uno per ogni host. I dati di utilizzo delle risorse nell'arco di tempo possono essere visti in base all'ultima ora, giorno, mese o anno. Tutte le informazioni e i grafici sono generati on-the-fly, facendo il parsing dell'albero XML ottenuto localmente dal demone "gmetad", vedi fig. 4.

Figura 4 : Sistema di monitoraggio: Ganglia

2.4.4 MRTG

MRTG[7] (Multi Router Traffic Grapher), applicazione con licenza GPL, scritta in PERL sia per Unix/Linux che per Windows, è in grado di monitorare qualsiasi host remoto che abbia il supporto al protocollo SNMP abilitato e conseguentemente, costruire una pagina web contenente i relativi grafici. Questo tool è stato installato nel server principale e viene usato principalmente per monitorare e tenere sotto controllo le

seguenti informazioni: temperatura ambiente nella sala del cluster, tensione fornita al cluster dall'UPS, traffico di rete relativo all'uplink della tratta GARR riservato al polo cyberSAR di ingegneria (switch CISCO 3750), carico determinato dai jobs in esecuzione sulle code riservate a CyberSAR. Le informazioni vengono acquisite eseguendo le seguenti operazioni:

- il demone "mrtg", interroga l'host remoto per chiedere il valore di uno specifico SNMP OID;
- viene aggiornata l'immagine del grafico con l'aggiunta di nuovi valori e cancellando quelli vecchi. I grafici sono immagini in formato png e le nuove variazioni possono essere memorizzate localmente o su un server di storage remoto dedicato;
- vengono memorizzati i nuovi valori in un file di log. I file di log possono essere memorizzati localmente o su un server di storage remoto dedicato;

I requisiti e componenti per la sua installazione sono:

- pacchetti mrtg: possono essere scaricati dal sito "http://oss.oetiker.ch/mrtg/". In relazione al tipo di file scaricato, potrebbe non essere richiesta una installazione specifica ma solo la loro decompressione nella directory di destinazione;
- interprete PERL;
- pacchetti "snmp" e "snmp-utils": includono diverse librerie e tools per settare o richiedere informazioni tramite gli SNMP agents e generare e gestire le SNMP traps;
- web server: utilizzato per visualizzare i grafici MRTG tramite browser, vedi fig. 5;
- esecuzione periodica del tool "mrtg" ad esempio tramite il comando "cron".

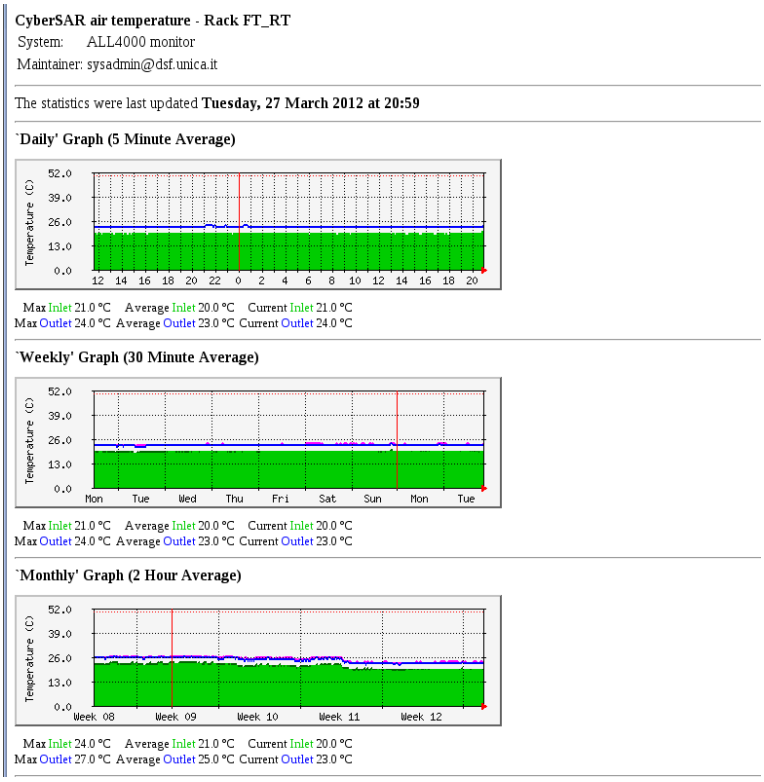

Figura 5 Grafico dati generato da mrtg

Per quanto riguarda la sua configurazione, mrtg necessita di un file ".cfg", che

può essere globale per tutti gli hosts, in cui sono definiti gli SNMP OIDs per ogni attività che si intende monitorare nell'host di destinazione.

3. CALCOLO DISTRIBUITO E GRID

Sebbene la potenza dei calcolatori vada aumentando di anno in anno, la necessità di capacità di calcolo risulta spesso maggiore delle risorse computazionali a disposizione. L'idea di base del calcolo distribuito è quella di unire le risorse di tanti computer che comunicano attraverso una rete, per realizzare potenze di calcolo enormi. Per la realizzazione di tali infrastrutture vengono usate varie architetture hardware e software, di seguito viene illustrata nel dettaglio la tecnica relativa al Grid computing.

3.1 Cluster Computing

Per cluster si intende un insieme di calcolatori, detti nodi, interconnessi tramite rete telematica, che vengono visti dall'utente come un unico grande server virtuale. Un cluster, è detto omogeneo quando è formato da nodi con identiche caratteristiche quali: l'architettura, la tipologia di hardware, il sistema operativo, etc. E' invece detto eterogeneo quando i nodi presentano caratteristiche diverse tra loro. Le motivazioni che spingono a realizzare soluzioni basate su cluster, sono le seguenti:

- Aggregazione della potenza: nel caso dovessero servire maggiori risorse, si aggiungono nodi al cluster e le prestazioni in termini di capacità computazionale aumentano;
- Incremento dell'affidabilità: in caso di mal funzionamento di uno o più nodi, i servizi forniti possono "migrare" su altri nodi del cluster, o essere ridistribuiti su altri server, in modo del tutto trasparente agli utenti. Ciò permette anche di fare manutenzione sui nodi senza dover fermare i servizi attivi ospitati;
- Riduzione dei costi: è possibile utilizzare calcolatori comuni per realizzare complesse soluzioni che altrimenti richiederebbero l'acquisto di costosi supercomputer commerciali.

I cluster per il calcolo parallelo, sono utilizzati in tutti quegli ambiti in cui è necessaria una grande potenza di calcolo resa disponibile grazie alla condivisione dell'hardware di elaborazione. Un'elaborazione, infatti, viene inviata al cluster che, lavorando come un'unica entità, ridistribuisce e suddivide il lavoro tra le macchine presenti nel cluster ottenendo una velocità nel calcolo superiore a quella che si otterrebbe su una singola macchina. In generale viene definito "parallelo" il calcolo eseguito in contemporanea da più processori, mentre "distribuito" quello in cui il programma viene diviso su più processori.

3.1.1 Concetto di Single System Image (SSI) e tassonomia di Flynn

Una delle caratteristiche comuni a tutti i cluster, è quella di cercare di implementare al meglio il concetto di SSI (Single System Image). SSI, è la capacità di fornire all'utente la visione di un unico grande computer, rendendo trasparente la presenza dei singoli nodi

individuali. L'utente non sa che l'elaborazione da lui richiesta, se disponibile, verrà messa in esecuzione su un particolare nodo del cluster. Sarà compito del cluster stesso, bilanciare le risorse disponibili in quel momento e distribuire il carico di lavoro in modo adattivo e dinamico. L'idea di parallelizzare i calcoli per la risoluzione di un problema, è abbastanza datata considerando la storia relativamente breve dei sistemi di calcolo. Risale infatti, al 1967 l'articolo nel quale Gene Amdahl, gettando quelle che sono considerate le basi del calcolo parallelo, introduce la legge utilizzata ancora oggi per predire il massimo aumento prestazionale teorico ottenibile aumentando il numero di processori. Ancora prima, nel 1966 Michael J. Flynn classificava, le architetture dei sistemi di calcolo secondo la capacità di elaborare contemporaneamente flussi multipli di istruzioni o dati. La "Tassonomia di Flynn", attualmente valida, è in grado di ripartire tutti i sistemi di calcolo in una delle seguenti categorie:

- **SISD** (Single Instruction Single Data): le istruzioni vengono eseguite sequenzialmente su un dato per volta. Fanno parte di questa categoria tutte le macchine a registri con architettura di von Neumann e quindi la maggior parte degli elaboratori ad uso privato oggi in commercio. La maggior parte di questi, è tuttavia in grado di eseguire più istruzioni in maniera concorrente tramite l'utilizzo di tecniche sofisticate per la ripartizione del tempo d'utilizzo dell'unità di elaborazione tra più sequenze di istruzioni, come ad esempio l'uso del multi-threading e delle pipeline;
- **SIMD** (Single Instruction Multiple Data): ogni singola istruzione di tali sistemi opera su array di dati: "processori vettoriali". Mentre in precedenza il calcolo vettoriale era prerogativa esclusiva dei super computer, al giorno d'oggi la maggior parte dei microprocessori multi-purpose commerciali contiene sottoinsiemi di istruzioni che operano su più dati contemporaneamente (ad esempio le istruzioni SSEx dei processori x86 Intel), così come tutte le moderne schede grafiche sono equipaggiate con GPU (Graphical Processing Unit) capaci di calcoli vettoriali e matriciali per l'utilizzo in applicazioni grafiche. I sistemi capaci di calcolo vettoriale implementano il così detto parallelismo sui dati poiché capaci di operare su più dati contemporaneamente. A dispetto della loro efficienza nell'esecuzione di algoritmi che operano su vettori di dati, le architetture di tipo SIMD sono spesso molto complesse e necessitano di software scritto ad-hoc;
- **MISD** (Multiple Instruction Single Data): più istruzioni diverse vengono eseguite contemporaneamente sullo stesso dato. Tipicamente nella maggior parte dei problemi si ha a che fare con grandi quantità di dati perciò le architetture di tipo SIMD e MIMD sono state impiegate in misura di gran lunga maggiore rispetto a quelle MISD. Non esistono dunque sistemi commerciali di questa tipologia che hanno conosciuto una rilevante diffusione;
- **MIMD** (Multiple Instruction Multiple Data): sono in grado di eseguire contemporaneamente più istruzioni diverse su dati diversi poiché avendo a disposizione molteplici unità di elaborazione indipendenti tra loro. Trovano il loro naturale impiego nel calcolo parallelo o in applicazioni che beneficiano del parallelismo raggiungendo le più alte prestazioni fino ad ora ottenute.

Le categorie SIMD e MIMD, rappresentano le tipologie sulle quali oggi maggiormente si concentra l'attenzione della ricerca e del mercato. La varietà di sistemi MIMD è tale da poter introdurre un'ulteriore classificazione in base all'accesso alla memoria da parte delle unità di elaborazione:

- **MIMD a memoria condivisa:** tutti i processori che sono contenuti nel sistema condividono la stessa memoria per mezzo di hardware dedicato. Le macchine che contengono da 2 a 64 unità di elaborazione indicate col nome di SMP (Simmetric Multi Processing) fanno parte di questa categoria. Gli elaboratori SMP possono essere a loro volta distinti in base a come i vari processori che contengono accedono alla memoria principale. Le architetture che consentono ai processori un accesso paritetico a tutta la memoria prendono il nome di UMA (Uniform Memory Access). Tuttavia la condivisione di tutta la memoria può rappresentare un potenziale "collo di bottiglia" quando i processori concorrono per l'accesso ad essa. Nelle architetture di tipo NUMA (Non-Uniform Memory Access) invece, ogni processore, pur mantenendo una visione globale dell'intera memoria del sistema, ha una limitata area di memoria locale alla quale accede esclusivamente e con velocità maggiore, mentre si ha una perdita di prestazioni solo nel caso di accessi ad aree di memoria comuni, o comunque non locali. In questi casi l'accesso deve essere regolato da complessi meccanismi necessari per mantenere coerente lo stato della cache di ogni processore. Ciò comporta una crescita della complessità dell'hardware al crescere del numero di processori. Per essere realmente utilizzabili le architetture di tipo NUMA dipendono pesantemente da tali meccanismi per mantenere la coerenza, infatti ci si riferisce a tali architetture con l'acronimo ccNUMA (cache-coherent NUMA);
- **MIMD a memoria distribuita:** sistemi che possiedono un'area di memoria separata e non direttamente accessibile da parte di altri nodi. I più potenti sistemi di calcolo oggi utilizzati appartengono a questa categoria e ci si riferisce ad essi con l'acronimo MPP (Massively Parallel Processing). Negli MPP ogni nodo contiene una o più unità di elaborazione e aree di memoria ad essi associate. Ogni nodo è connesso agli altri con topologie spesso complesse, volte ad ottimizzare costi, ampiezza di banda e latenza. L'efficienza dell'interconnessione tra i nodi è cruciale per le prestazioni dell'intero sistema poiché non essendoci porzioni di memoria condivisa, il passaggio di dati tra le unità di elaborazione può avvenire solo attraverso lo scambio di messaggi. Tutti in nodi di elaborazione affiancati da altri nodi specializzati per le operazioni di input/output contribuiscono a formare un unico grande calcolatore. Oltre ai sistemi MPP, la tipologia di sistemi MIMD a memoria distribuita che negli ultimi anni ha ricevuto maggiori attenzioni da parte delle comunità scientifiche e dai produttori di hardware è rappresentata dai cluster di computer. Spesso i nodi degli MPP fanno uso di unità di elaborazione di tipo SIMD in modo da aumentare ulteriormente il grado di parallelismo. Similmente agli MPP anche i cluster di computer sono costituiti da più nodi, ma a differenza dei primi, ogni nodo è costituito a sua volta da un sistema indipendente che comprende unità di elaborazione, memoria e periferiche di input/output.

3.1.2 Meccanismi di comunicazione tra processi o tra processori

L'idea base del calcolo parallelo si fonda sul principio che diversi processori cooperano nella soluzione di un singolo problema. Ma per ottenere un calcolo parallelo efficiente, non basta semplicemente mettere più processori uno a fianco all'altro e connetterli ad una velocità sufficiente. Un programma che viene svolto correttamente da una singola CPU può presentare seri problemi se svolto in parallelo. E' stato quindi necessario riprogettare gli algoritmi che funzionavano correttamente in un'unica cpu in modo che quando vengono aggiunti altri processori, alcuni calcoli possano essere eseguiti sfruttando le pipeline software. Questo metodo divide il lavoro come in una catena di montaggio: se il calcolo può essere diviso in n stadi diversi e significativi, possono essere usati con efficienza n processori.

Se però uno stadio è più lento degli altri, questo rallenterà tutto il sistema. Alla luce di queste considerazioni la maggior parte degli algoritmi, deve essere riscritta per sfruttare l'hardware parallelo. Per ottimizzare il calcolo parallelo, sono stati sviluppati moltissimi software sia a livello di sistema operativo che a livello di linguaggio di programmazione. Questi sistemi devono contenere i meccanismi necessari a suddividere il calcolo tra le unità. Essi possono supportare:

- un parallelismo implicito in cui il sistema suddivide il problema e lo fornisce ai processori automaticamente;
- un parallelismo esplicito in cui il programmatore deve inserire degli indici per suddividere il proprio codice.

I processori possono comunicare tra loro per risolvere un problema in modo coordinato, oppure funzionare in maniera totalmente indipendente, a volte sotto il controllo di un processore che fornisce loro i dati e ne preleva i risultati. E' necessario disporre di meccanismi di comunicazione tra processi o tra processori. In linea generale i sistemi multiprocessore possono seguire due modelli base di organizzazione. Nei sistemi multi processore a bus singolo, vedi fig. 1, il mezzo di collegamento è unico ed è collocato tra i processori e la memoria. Il bus è usato per tutti gli accessi in memoria e la comunicazione è effettuata mediante un ampio spazio d'indirizzamento condiviso. Occorre quindi gestire i "conflitti" per l'accesso alla memoria e sincronizzare gli accessi alla stessa locazione da parte di più processori.

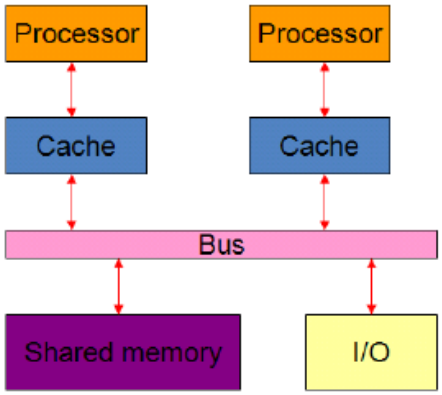

Figura 1 Sistemi multiprocessore a bus singolo

Nei sistemi multiprocessore connessi via rete, vedi fig. 2, invece la memoria è annessa a ciascun processore. Il sistema d'interconnessione è coinvolto solo nella comunicazione fra processori diversi. Ogni nodo accede direttamente alla propria parte di memoria e l'informazione condivisa deve essere replicata nelle memorie dei diversi nodi.

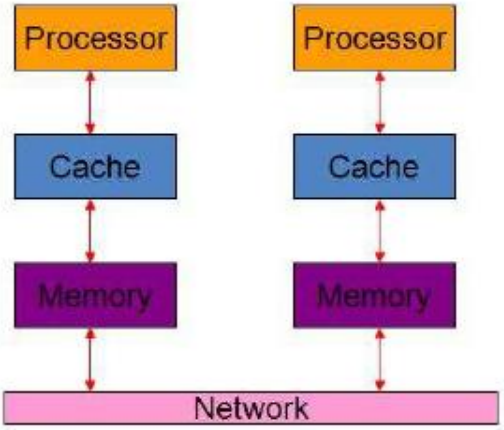

Figura 2 Sistemi multiprocessore connessi via rete

Un aspetto fondamentale di questo argomento è quindi la gestione della comunicazione tra i processi IPC (Inter Process Communication services) e i servizi messi a disposizione dal sistema operativo. Essi permettono ai vari programmi che sono eseguiti su di una stessa macchina di condividere in maniera collaborativa alcune risorse e di accordarsi sulla priorità di accesso, che possono essere usate da un solo processo alla volta. Tali comunicazioni sono realizzate mediante strutture dati rese disponibili dal kernel del sistema operativo. Ciascuna di queste strutture dati è indicata da un identificatore, avente lo stesso significato dell'identificatore di file aperto, mediante il quale i processi possono acquisire, utilizzare e rimuovere le strutture. Tale identificatore viene ottenuto da una system call, specificando alcuni parametri per permettere ad insiemi di processi diversi di condividere strutture diverse. Sono disponibili tre tipologie di comunicazioni tra processi:

- **code di messaggi:** permettono a due o più processi di scambiarsi messaggi in maniera asincrona. Il sistema operativo funziona quindi come deposito, in cui i processi immettono o prelevano messaggi;
- **semafori:** fanno sì che un processo possa accedere in maniera esclusiva a una certa risorsa condivisa. Quando una risorsa deve essere usata da un solo processo alla volta, è necessario associare un semaforo a tale risorsa e assicurarsi che tutti i processi che accedono a tale risorsa ottengano il via libera dal semaforo prima di cominciare a usarla;
- **memoria condivisa:** consente di far condividere a più processi una stessa zona di memoria. Questo non sarebbe possibile senza l'intervento del kernel, perché nei moderni sistemi operativi i processi hanno un proprio spazio di indirizzamento e quindi non possono accedere, per motivi di sicurezza, direttamente alla memoria di altri processi.

3.1.3 Ambienti d'esecuzione

La disponibilità di software standard per la programmazione e l'esecuzione di applicazioni parallele ha aiutato notevolmente la diffusione dei cluster di computer. Un'applicazione parallela esegue un programma il cui carico computazionale può essere suddiviso tra i vari nodi, consentendo vantaggi prestazionali proporzionali al grado di parallelismo del programma stesso. L'ambiente di esecuzione per le applicazioni parallele è realizzato tramite l'impiego di librerie che "parallelizzano" i programmi. Tali librerie, che consentono di sfruttare il paradigma di programmazione message passing, forniscono meccanismi per

formare canali di comunicazione e consentire il passaggio di messaggi esplicito tra i processi che compongono le applicazioni parallele. Le specifiche di Message Passing Interface (MPI) costituiscono ormai uno standard de facto per la programmazione di applicazioni parallele in sistemi a memoria distribuita come i cluster. Esistono molte implementazioni di librerie MPI che sono ormai disponibili per un gran numero di architetture hardware ed interfacce verso tutti i linguaggi di programmazione più utilizzati. Insieme alle librerie MPI, le librerie di message passing più utilizzate sono le librerie Parallel Virtual Machine (PVM).

3.2 Grid Computing

Il termine Grid Computing[23][29] è stato coniato nel 1990 per indicare una infrastruttura avanzata di calcolo distribuito utilizzata in ambito scientifico e ingegneristico. Questo termine può essere letteralmente tradotto in Italiano con le parole “calcolo a griglia” e sta ad indicare un paradigma del calcolo distribuito, costituito da un'infrastruttura altamente decentralizzata e di natura variegata, in grado di consentire ad un vasto numero di utenti l'utilizzo di risorse, prevalentemente CPU e storage, provenienti da un numero non definito di calcolatori interconnessi da una rete. Questo concetto, si avvicina fortemente a quello di ”electric power Grid”, modello delle reti di distribuzione dell’energia elettrica, al quale idealmente si ispira. Le griglie computazionali sono state ideate con l’obbiettivo di realizzare una infrastruttura di calcolo mondiale, che dia la possibilità agli utenti di poter reperire risorse di calcolo con la stessa facilità con la quale oggi si può usufruire dell'energia elettrica. Questa ambiziosa e interessante idea è stata proposta nel 1969 da Leonard Kleinrock. Esistono numerose definizioni di Grid computing fornite da studiosi e ricercatori, esse si possono riassumere nelle seguenti:

- un sistema che coordina risorse non soggette ad un controllo centralizzato. Questo significa poter integrare e coordinare risorse e utenti che non sono definiti in un unico dominio di gestione;
- un sistema che utilizza interfacce e protocolli standard, aperti e general purpose;
- un sistema che produce una qualità di servizio accettabile e che nel suo complesso è più efficiente rispetto alla somma delle sue singole parti.

Le Grid sono in grado di fornire agli utenti di un gruppo senza una particolare caratterizzazione geografica ne tanto meno istituzionale, la potenzialità di accedere alla capacità di calcolo e di memoria di un sistema distribuito, garantendo un accesso coordinato e controllato alle risorse condivise come se fosse un unico sistema di calcolo logico cui sottomettere i propri job. Questo tipo di architettura si presta molto bene alla risoluzione di problemi con insiemi di dati molto grandi, ingestibili da un singolo sistema di calcolo e ad elevato grado di parallelismo. Questo tipo di problemi è divisibile facilmente in task indipendenti l’uno dall’altro, che non hanno bisogno di alte prestazioni nello scambio di messaggi. Il problema dell’alta latenza nella comunicazione è in definitiva l’unica limitazione delle Grid che per contro, possono essere espanse in termini di potenza di calcolo e capacità di storage virtualmente all’infinito, senza incorrere nei problemi tipici dei sistemi centralizzati. I vantaggi che si hanno utilizzando un’architettura Grid vanno ben oltre dal semplice aumento della potenza di calcolo, infatti, essa permette di:

- integrare e coordinare risorse non sempre appartenenti allo stesso ambito, stabilendo un insieme di regole e permessi di condivisione garantite da procedure di autenticazione e autorizzazione;

- gestire le risorse in modo trasparente in quanto agli utenti è permesso accedere alle risorse remote come se fossero locali;
- garantire servizi in termini di performance, sicurezza, tolleranza agli errori, disponibilità e tempi di risposta;
- aprire sessioni in qualsiasi macchina, agli utenti autorizzati ovunque essi si trovino, in quanto è una tecnologia disponibile ovunque.

3.2.1 Campi di applicazione del Grid Computing

Oggi, i computer sono utilizzati per modellare e simulare problemi scientifici ed ingegneristici complessi, diagnosticare analisi mediche, controllare equipaggiamenti industriali, studiare e prevedere le condizioni meteo, eseguire operazioni bancarie, biotecnologie e molto altro. L'osservazione principale è che la media degli ambienti computazionali esistenti risulta inadeguata ad assolvere tali funzioni al fine di ottenere obiettivi così sofisticati e complessi. Le Grid possono risolvere questo problema, integrando risorse di strumentazione, di visualizzazione, di calcolo e di informazione, provenienti da domini amministrativi diversi e geograficamente distribuiti. Attualmente, questa tecnologia che in passato era utilizzata maggiormente in ambito scientifico, si è rivelata molto efficiente anche per grosse multinazionali, laboratori privati, organi governativi, enti pubblici che possono così beneficiare della conseguente capacità di aggregazione e accesso a risorse distribuite. Durante le varie sperimentazioni si è visto che un modello Grid può essere applicato in modo efficiente a diverse applicazioni come:

- Supercalcolo Distribuito: offre la possibilità di poter utilizzare più supercomputer o cluster garantendo di superare le limitazioni di calcolo esistenti. Questo tipo di applicazioni possono utilizzare un sistema Grid per unificare le risorse di calcolo di alcuni, o molti supercomputer, al fine di risolvere problemi che, altrimenti, sarebbe oggi proibitivo risolvere su di un unico calcolatore, per quanto potente;
- On Demand Computing: le applicazioni on demand utilizzano sistemi Grid per rendere disponibili risorse di calcolo che, per il loro saltuario utilizzo, non sarebbe conveniente avere sempre a disposizione. Queste risorse possono essere processori, software, archivi di dati, strumenti molto specializzati e così via. Al contrario del supercalcolo distribuito, queste applicazioni sono spesso motivate dal rapporto costo-prestazioni, piuttosto che dalla necessità di performance;
- Data Intensive Computing: nelle applicazioni di questo tipo, lo scopo principale è la gestione di enormi quantità di dati distribuiti geograficamente. In questi sistemi si riscontrano spesso problemi legati all'alto carico computazionale e alle modalità di trasferimento di tali moli di dati;
- Calcolo Collaborativo: questo tipo di applicazioni mirano soprattutto a favorire le comunicazioni e le collaborazioni tra le persone fisiche, pertanto sono spesso pensate in termini di spazi virtuali. Molte di esse devono rendere disponibili risorse di calcolo condivise, ad esempio archivi di dati, e pertanto condividono tutti gli aspetti delle applicazioni già analizzate. In questa situazione, ciò che assume particolare rilievo è la necessità di fornire queste funzionalità in tempo reale.

A proposito del campo applicativo, il sistema di Grid computing realizzato dal consorzio Cybersar, è un'infrastruttura capace di fornire calcolo distribuito intensivo e ad alte prestazioni a comunità internazionali disperse geograficamente, ma cooperanti per uno stesso obiettivo.

Ciò si traduce nell'accesso continuo ed organizzato ad ingenti risorse di vario tipo quali calcolatori, banche dati, librerie software, sparse su scala geografica, ma interconnesse fra loro tramite reti di comunicazione dati ad alta velocità. L'enorme vantaggio di questa Grid sta nel fatto che essa rende disponibili grandi risorse di calcolo e dati che nessuna singola organizzazione sarebbe altrimenti in grado di acquistare e gestire da sola. Ciò ha permesso di ottimizzare l'utilizzo, la gestione ed i costi delle risorse. Non è più necessario avere singoli sistemi di calcolo per ogni gruppo di ricerca ma tutti possono accedere alla stessa infrastruttura in modo ottimizzato. Il risultato concreto di tale iniziativa è efficacemente rappresentato dai 1400 processori che operano e che sono capaci di fornire circa 100 volte la potenza di calcolo disponibile per un singolo gruppo di ricerca e permettono quindi di realizzare in pochi giorni calcoli complessi che altrimenti richiederebbero mesi. Grazie ad alcune scelte strategiche è stato possibile implementare e rendere disponibili servizi basati su standard di comunicazione come LCG (LHC Computing Grid) e gLite del progetto EGEE (Enabling Grids for E-sciencE). Ciò ha reso possibile la partecipazione di CyberSAR, a livello di infrastruttura di calcolo, a progetti quali IGI[8], vedi fig. 3, e a esperimenti come ALICE[9] (A Large Ion Collider Experiment).

Figura 3 Italian Grid Infrastructure

3.2.2 Organizzazione gerarchica della Grid

Secondo il modello Grid nello standard EGEE, ci sono diverse strutture organizzate in modo gerarchico, vedi fig. 4, che hanno il compito di gestire e controllare il funzionamento dell'intera Grid stessa. Il livello più alto di questa scala gerarchica è rappresentato dall'OMC (Operation Manager Centre), esso è situato nelle sedi del CERN di Ginevra ed è costituito da un esperto gruppo di lavoro con notevoli esperienze nell'infrastruttura Grid. Ha responsabilità generiche di coordinamento e si occupa di verificarne il corretto funzionamento, definendo la politica operativa che deve essere attuata dai CIC (Core Infrastructure Centre) e dai ROC (Regional Operation Centre). I CIC, hanno il ruolo di far funzionare l'infrastruttura essenziale

di Grid offrendo i servizi infrastrutturali essenziali in ogni singolo sito. Di questi centri ne esistono quattro e sono situati al CERN, nel Regno Unito, in Italia e in Francia. I CIC svolgono anche il compito di supportare i ROC nella risoluzione di problemi. Uno dei CIC a rotazione settimanale è responsabile di mantenere un operatore, 24/24 ore e 7/7 giorni, che controlla l'infrastruttura Grid. I ROC, sono attualmente nove, hanno il ruolo di coordinare e supportare, all'interno della nazione di riferimento, sia l'implementazione di nuovi siti che di nuove versioni di middleware. Il CERN è ROC per tutte quelle nazioni in cui non ne sia già presente un altro. I Site Manager sono i responsabili locali di un sito Grid, hanno il compito della gestione, manutenzione hardware e dell'installazione e rimozioni dei nodi. Essi si occupano inoltre di risolvere eventuali problemi di rete e di gestire e stabilire le politiche di accesso alle risorse e agli utenti.

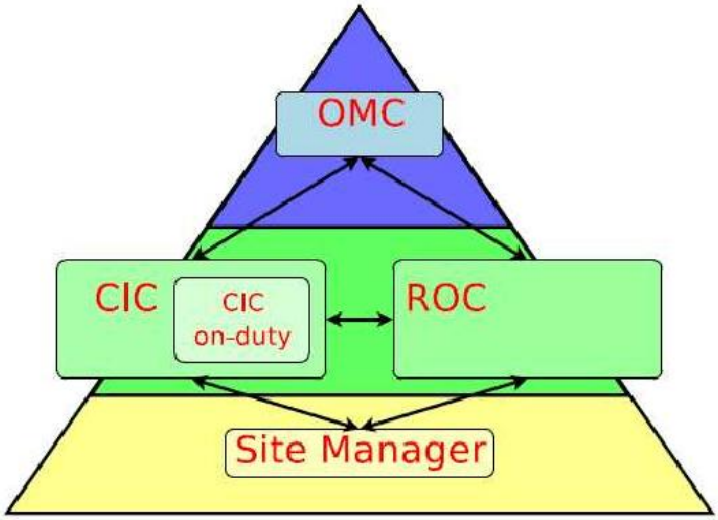

Figura 4 Struttura della gestione Grid

L'architettura di una Grid può essere suddivisa su quattro livelli, vedi fig. 5:

- livello di rete: rappresenta gli apparati che garantiscono la connettività a tutte le componenti del livello superiore, possono essere sia entità logiche che fisiche;
- livello delle risorse: si divide a sua volta in due sottolivelli. Il livello delle risorse si occupa di gestire le risorse di calcolo, storage o altro che afferisce alla Grid. In esso vengono definiti i protocolli API e SDK che si occupano del controllo e del lancio di operazioni sulle risorse. Il livello della connettività in cui sono definiti i protocolli di comunicazione e autenticazione per le transazioni di rete e lo scambio dei dati con il livello inferiore;
- livello collettivo: relativo al middleware, fornisce funzioni di RMS (Resource Management System) e alta affidabilità. Come per i cluster il middleware è di fondamentale importanza per la costituzione della Grid. In esso sono contenuti i servizi API utilizzati per la gestione e la condivisione delle risorse e mostrano all'utente l'allocazione delle risorse di cui ha bisogno;
- livello applicativo: fornisce l'interfaccia con la quale l'utente può interagire con il sistema e contiene le applicazioni rivolte all'utente ed il meccanismo di gestione delle organizzazioni virtuali.

Figura 5 Livelli dell'architettura Grid

3.2.3 Le Organizzazioni virtuali VO e le VOMS

Gli enti che partecipano alla costituzione della Grid vengono dette Virtual Organizations (VO) e sottostanno a politiche di autorizzazione e autenticazione concordate insieme agli organi di coordinamento. Le organizzazioni virtuali sono molto varie tra di loro perché hanno strutture ed esigenze diverse. All'interno di ogni VO sono quindi definiti dei ruoli che permettono di assegnare agli utenti le priorità sull'uso delle risorse una volta eseguita l'autenticazione. Questo implica che gli utenti non sono tutti uguali, alcuni hanno dei privilegi e possono accedere a determinate risorse, altri invece possono avere delle priorità nell'esecuzione dei job e così via. Per questo motivo sono organizzate gerarchicamente e pensate per avere una struttura dinamica per rispondere alle esigenze di un numero variabile di utenti e risorse. Fra organizzazioni virtuali diverse, ci possono però essere relazioni e collaborazioni. Un utente quindi, può appartenere a più VO contemporaneamente. I server VOMS (VO Management Service) sono un sistema di gestione delle autorizzazioni Grid e gestiscono le informazioni riguardanti le regole e i privilegi degli utenti all'interno di una VO. Queste informazioni sono disponibili per ogni utente grazie alla creazione di un proxy in fase di autenticazione. Durante la creazione del proxy, uno o più server VOMS vengono contattati per l'acquisizione di un certificato chiamato AC (Attribute Certificate). Tale certificato è assegnato dalla VO e contiene le informazioni riguardanti i membri del gruppo e le regole associate alle VO stessa. Il proxy e la relativa AC rilasciata, tipicamente hanno una durata di dodici ore ma esiste la possibilità di prorogare questo arco di tempo.

3.2.4 gLite

Nell'infrastruttura Grid realizzata, è stato installato e configurato il middleware gLite costruito su Globus[10], molto usato nelle Grid poiché strutturato in maniera modulare con una architettura orientata ai servizi. Nasce da un progetto basato sui middleware EDG e LCG, ai quali aggiunge nuove funzionalità in tutti i livelli della struttura software. In particolare sono stati privilegiati gli aspetti che riguardano la sicurezza, la gestione dei dati e la sottomissione dei job. La creazione di un sistema enormemente scalabile, richiede dei

protocolli, in accordo a determinati standard, adatti a supportare un'ampia varietà di offerta di servizi. L'autorità in questo campo è costituita dalla Globus Alliance, un organismo che promuove l'utilizzo delle tecnologie Grid e sviluppa attivamente il pacchetto software Globus Toolkit, completamente open-source e protetto dalla Globus Toolkit Public License. Il tool si compone di quattro parti:

1. Gestione della Sicurezza: GSI (Grid Security Infrastructure) è un meccanismo di autenticazione che si occupa della sicurezza nell'ambiente Grid, garantendo l'accesso solamente agli utenti autorizzati. E' basato sullo standard dei certificati X.509, la crittografia asimmetrica e sul protocollo SSL;
2. Gestione delle risorse: GRAM (Globus Resource Allocation Manager) si occupa della gestione delle risorse ed ha il compito di abilitare un accesso sicuro e controllato alle risorse computazionali, gestendo l'esecuzione remota di operazioni sulle risorse stesse. Il Globus Toolkit, mette a disposizione una struttura a livelli per la gestione delle risorse computazionali in cui ai livelli più alti ci sono tutti i servizi relativi alla loro gestione, mentre nei livelli inferiori troviamo quelli per l'allocazione. Il GRAM rappresenta il livello più basso di questa struttura. Il gatekeeper è un processo del GRAM che gestisce la richiesta di un utente inviandola al job manager, il quale dopo aver creato un processo per la richiesta ne controlla l'esecuzione comunicandone lo stato all'utente remoto. Il protocollo usato dal GRAM è basato su una serie di RPC (Remote Procedure Call) e sul protocollo HTTP;
3. Gestione dei dati: LFC (LCG File Catalogue) è il sistema di data management della Grid. Esso si occupa della gestione dei dati e di fornire le informazioni che si riferiscono alla loro allocazione. E' interpellato anche quando viene effettuata una copia dei dati, perché possiede dei servizi per la gestione della replica e della comprensione dei meta-dati. Per spiegare meglio come funziona questo servizio è necessario conoscere come sono memorizzati, salvati e identificati i dati nelle Grid. I dati sono registrati in file, ai quali vengono assegnati quattro parametri:
   - GUID (Grid Unique Identifier): identifica logicamente un file in maniera univoca tramite una stringa di caratteri. E' creato da un particolare algoritmo che ne garantisce l'unicità. Un esempio di GUID è il seguente:

     guid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6
   - LFN (Logical File Name): è il nome assegnato dall'utente al file, a cui è associato il GUID. Un esempio di LNF può essere il seguente:

     lfn:/Grid/cms/20030203/run2/track100
   - TURL (Transport URL): fornisce le informazioni necessarie per il trasferimento del file;
   - SURL (Storage URL): chiamato anche PFN (Physical File Name), memorizza la locazione fisica del file includendo l'indirizzo del server che si occuperà della memorizzazione dei dati e il protocollo di accesso.

   Le tre componenti che svolgono le principali funzioni di gestione dei dati sono:

   - Gridftp: è un protocollo utilizzato per realizzare lo scambio di file tra le varie risorse all'interno della Grid. E' stato realizzato migliorando ed estendendo il protocollo FTP in modo da ottimizzarlo per le caratteristiche di questa struttura, permettendo così di aumentare la capacità e la sicurezza del trasferimento dati grazie all'utilizzo dei meccanismi definiti dal GSI;

- Globus Replica Catalog: al fine di massimizzare la velocità di accesso ai dati, questo sistema, permette la replicazione degli stessi su diversi storage, anche remoti;
- Globus Replica Manager: permette la replicazione e il trasferimento dei dati grazie alla cooperazione del protocollo Gridftp con il Globus Replica Catalog.

4. Gestione delle Informazioni: il servizio GIS (Grid Information Service) raggruppa le informazioni di stato delle varie risorse ed è suddiviso in tre principali servizi: MDS (Monitoring and Discovering Service), GIIS (Grid Index Information Service) e GRIS (Grid Resource Information Service). Il primo fornisce un servizio d'informazione creando un sistema di directory di servizi utilizzato per recuperare informazioni sullo stato delle risorse presenti. Ogni risorsa dispone di un software GRIS, che reperisce le informazioni sul proprio stato e le invia periodicamente a un server GIIS. Generalmente questo è attivo su una sola macchina del sito Grid e fa parte di una struttura gerarchica formata da più GIIS server, ognuno dei quali provvede a recuperare le informazioni ed a inviarle al GIIS di livello superiore fino ad arrivare a quello di livello massimo chiamato top MDS.

3.2.5 Management e Scheduling delle Risorse

Il Resource Management System (RMS), è di fondamentale importanza per l'uso efficiente delle risorse del cluster. Il compito dell'RMS è quello di distribuire su tutto il cluster le richieste di risorse hardware delle applicazioni in esecuzione. I software che svolgono funzioni di RMS, solitamente sono composti di due parti:

- Il resource management, raccoglie informazioni relative alla presenza e allo stato delle risorse, mantenendo un'immagine aggiornata di quelle disponibili nel cluster in base ai cambiamenti dovuti ad allocazioni, de-allocazioni o indisponibilità dipendenti da altre ragioni quali il malfunzionamento dell'hardware. Raccoglie e autentica, inoltre, le richieste di allocazione, creando le strutture dati necessarie all'identificazione dei processi delle applicazioni che ne richiedono l'utilizzo;
- Il resource scheduling, sulla base delle informazioni fornite dalla componente di management, ordina le richieste di utilizzo delle risorse in modo da minimizzare i tempi di attesa e gestire le situazioni di concorrenza. Solitamente lo scheduling delle richieste viene regolato per mezzo di code che possono essere gestite tramite politiche configurabili, in modo da ottimizzare l'utilizzo delle risorse e distribuirle in maniera equa tra le applicazioni.

L'architettura tipica delle applicazioni di tipo RMS è di tipo client-server. La parte client, mantiene aggiornate le informazioni relative al sistema che lo ospita e le comunica alla componente di management tramite un processo di sistema in esecuzione su ogni nodo che vuole condividere le proprie risorse. La componente di management, o parte server, riceve con tali informazioni tutti i cambiamenti di stato delle risorse che avvengono sui singoli nodi. L'utente, ha la possibilità di sottomette le sue richieste di esecuzione di un'applicazione, solitamente tramite linea di comando oppure attraverso una interfaccia che interagisce con il sistema di RMS. Le applicazioni che possono essere sottomesse per l'esecuzione, sono di tipo interattivo o più comunemente, di tipo batch, in questo caso vengono dette job. Nella sottomissione di un job, l'utente deve provvedere a fornire all'RMS solo i dettagli necessari per l'esecuzione dell'applicazione come: la posizione dell'eseguibile, i dati in ingresso, la destinazione dei dati in uscita e se necessarie le proprie credenziali per l'autenticazione e autorizzazione. Successivamente alla sottomissione, l'RMS prenderà in carico l'esecuzione del

job, astraendo l'utente dalla complessità dei livelli software e hardware sottostanti. Esistono numerosi software che svolgono funzioni di RMS, tra cui la famiglia di batch system PBS (Portable Batch System), a partire dalla quale sono stati sviluppati OpenPBS, TORQUE o PBS Professional. Accanto ad essi i più utilizzati sono Maui Cluster Scheduler, Condor, LSF (Load Sharing Facility), SGE (Sun Grid Engine) e molti altri.

3.2.6 Meccanismi di Sicurezza negli ambienti Grid

Grid Security Infrastructure (GSI) è il meccanismo sul quale si basano le procedure di autenticazione degli utenti e delle risorse in un ambiente Grid. Esso è pertanto la struttura fondamentale su cui si appoggiano tutte le altre. Ogni volta che due entità, vogliono comunicare utilizzando la Grid come infrastruttura, devono autenticarsi. Questo è necessario per garantire che solo gli utenti autorizzati possano utilizzare le risorse di calcolo condivise, certificando la propria identità. GSI è basato su crittografia a chiave pubblica, certificati X.509 e sul protocollo di comunicazione SSL (Secure Socket Layer). Non è però, strettamente necessario che le comunicazioni siano sempre crittografate, poiché tale attività comporta un notevole aumento delle dimensioni dei dati, spesso inutile, come nel caso di dati grezzi di esperimenti o test. La sua implementazione Globus aderisce a GSS-API (Generic Security Service API), lo standard API per la sicurezza dei sistemi promosso dall'Internet Engineering Task Force (IETF). In questo standard sono definite le specifiche necessarie per un'autenticazione basata sul sistema dei certificati, detti anche Digital ID. Essi sono l'equivalente elettronico di un passaporto e in pratica servono a provare che qualcuno sia effettivamente chi sostiene di essere. I certificati vengono rilasciati da autorità apposite, chiamate CA (Certification Authority) che hanno il compito di verificare, al momento del rilascio del certificato, la veridicità delle informazioni contenute, cioè l'identità della persona alla quale il certificato viene rilasciato. La CA riceve dall'utente, generalmente via E-Mail, una copia della chiave pubblica, generata con un apposito software che viene utilizzato per compilare il certificato. Il sistema di autenticazione in Globus ha dovuto risolvere il problema di fornire un sistema di autenticazione di accesso controllato alle risorse e integrità dei dati; in particolare:

- one time login: un utente deve autenticarsi un'unica volta ad esempio all'inizio della giornata o al momento del lancio del job e poter lanciare programmi che acquisiscono risorse in tutta la Grid senza dover ripetere la procedura entro un certo arco di tempo;
- protezione delle credenziali: le credenziali personali (password o chiavi private) dell'utente devono rimanere al sicuro in ogni momento;
- possibilità di esportare il codice: è necessario che il codice utilizzato non vada in contrasto con le legislazioni dei paesi che collaboreranno ai vari progetti;
- sistema di credenziali/certificazione uniforme: è necessario utilizzare uno standard comune, almeno per il sistema di mutua identificazione tra domini diversi (X.509 per le credenziali e il software SSLeay per la certificazione).

I primi due punti sono stati risolti tramite l'introduzione del concetto di user proxy. Al momento del login l'utente provvede a creare il suo proxy attraverso un certificato X.509 firmato elettronicamente da lui stesso. L'user proxy disporrà quindi di una chiave privata ed una pubblica. Quella privata rimarrà custodita all'interno della macchina che ha creato il proxy, mentre quella pubblica sarà contenuta nel certificato firmato dall'utente. Infine, il certificato del proxy conterrà, oltre alla chiave pubblica firmata dall'utente, anche l'intervallo

di tempo per cui tale certificato avrà validità e ulteriori informazioni che si riferiscono ai permessi del proxy.

3.2.7 Servizi della Grid

Una tipica architettura Grid è composta da diversi elementi, vedi fig. 6, ognuno con specifici compiti, collocabili rispetto al sito a livello globale o locale. Gli elementi di livello globale sono:

- Resource Broker (RB): è la macchina che si occupa di assegnare ai job le risorse disponibili in base alle specifiche espresse nella descrizione del job. Se non è specificata una risorsa su cui eseguire il job, l'RB ne sceglie una tra quelle disponibili;
- Logging & Bookkeeping (LB): è la macchina che contiene tutte le informazioni sullo stato di tutti i processi di tutti gli utenti, memorizzando in opportune tabelle sia le azioni degli utenti che lo stato dei job. L'utente può interagire con essa per conoscere lo stato dei job usando opportuni comandi;
- Berkley Database Information Index (BDII): è il database utilizzato per memorizzare le informazioni che riguardano le risorse. Tiene in memoria una serie di tabelle per fornire le informazioni che riguardano il tipo di hardware, sistema operativo e lo stato di una risorsa. Una struttura Grid con più siti, deve avere un BDII per ogni sito e un BDII globale che prende il nome di top BDII che si occupa di memorizzare le risorse di tutti i siti;
- User Interface (UI): consente l'accesso agli utenti nell'infrastruttura Grid e l'esecuzione delle operazioni di base come la visualizzazione delle risorse disponibili, il recupero delle informazioni sul loro stato, la gestione ed il monitoraggio sullo stato dei job ed il recupero dell'output al termine dell'esecuzione.

Gli elementi di livello locale sono:

- Computing Element (CE): è composto dal gatekeeper e dallo scheduler che gestiscono la sottomissione dei job ai nodi di calcolo. Una volta che un utente lancia il comando per eseguire un job dalla UI, l'RB deciderà quale CE sarà responsabile per la sua esecuzione e a sua volta, sceglierà il nodo più adatto all'esecuzione del job. Il criterio di scelta è basato su algoritmi che ottimizzano l'utilizzo delle risorse;
- Storage Element (SE): è una macchina che si occupa della memorizzazione dei dati, del loro accesso e della loro replica. L'utenza che utilizza lo storage lo fa attraverso le UI che a loro volta si rivolgono all'SE attraverso meccanismi che gestiscono automaticamente la gerarchia di file e directory, permettendo l'accesso ai dati. Gli SE controllano un disk server che contiene gli array del disco e un sistema di controllo che prende il nome di Mass Storage System (MSS). Il Mass Storage System è composto da un gestore gerarchico della memoria che prende il nome di Hierarchical Storage Management (HSM) e si occupa di effettuare la migrazione tra i file di front-end e back-end. Esistono diverse tipologie di SE che possono essere raggruppate in quattro grandi categorie:
    - Classic SE: è un architettura pensata con un singolo server GridFTP e con una sola area di memoria del disco. Il server supporta un trasferimento dati sicuro;
    - Disk Pool Manager (DPM): viene molto utilizzato per creare storage element di piccole dimensioni con le funzioni base di storage. Esso è composto da un server che fornisce un singolo punto di accesso verso un pool disk Server;

  - dCache: è stato progettato in modo da poter avere uno Storage Element che possiede sia un MSS che un pool disk con un array su larga scala che gestisce lo Storage System. Esso è strutturato con un nodo server e un pool di uno o più nodi da gestire. Il server rappresenta il punto di accesso allo Storage Element dove è presente un file system virtuale con struttura ad albero che memorizza le informazioni relative ai vari pool disk. I nodi gestiti attraverso dCache possono essere aggiunti al pool in modo dinamico;
  - Castor: è pensato per gestire Storage Element su larga scala, usando MSS con nastri di memoria che contengono le informazioni front-end e back-end. Questa soluzione nasconde al suo interno un complesso protocollo di accesso e trasferimento.
- Worker Node (WN): sono i nodi più diffusi in un sito Grid, il loro compito è quello di eseguire i job sottomessi dagli utenti.

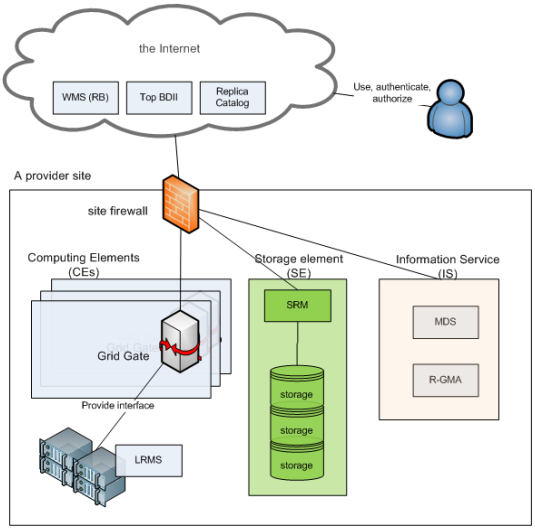

Figura 6 Architettura di gLite

3.2.8 Fasi di sottomissione di un Job

Quando un utente esegue il comando per la sottomissione di un job, vengono svolte diverse fasi in cui le componenti Grid interagiscono tra loro, vedi fig. 7:

- Fase 1: l'utente accede al suo account personale nella UI e crea un certificato proxy, che gli consente di utilizzare le risorse della Grid. Il job per essere sottomesso deve essere descritto utilizzando il linguaggio JDL (Job Description Language) usato dall'RB per verificare l'esistenza delle risorse necessarie per la sua esecuzione. LB che monitora lo stato del job, crea i file di log relativi ad esso e imposta il suo stato in “SUBMITTED”;
- Fase 2: l'RB verifica che le risorse richieste siano disponibili interrogando il BDII. Questa componente Grid tiene conto anche delle eventuali politiche e restrizioni espresse dall’amministratore rispetto all'utente. Il servizio che permette tale controllo è l'ISM (Information Super Market), che facendo riferimento ad una cache interna, tiene in memoria le informazioni aggiornate dal BDII e consente di determinare quale CE dispone delle risorse necessarie. Infine, l'RB consulta l'LFC per determinare la locazione di

eventuali file d'input. Durante questa fase LB aggiorna lo stato del job in "WAITING";

- Fase 3: l'RB prepara il job per la sottomissione, creando un apposito script con altri parametri passandolo al CE selezionato nella fase precedente. L'LB cambia lo stato del job in "READY";
- Fase 4: il CE riceve lo script di sottomissione del job e le invia al job manager per l'esecuzione, in modo da decidere su quale WN sottometterlo. Lo stato del job cambia in "SCHEDULED";
- Fase 5: il job manager manda il job al WN prescelto per l'esecuzione, lo stato del job diventa "RUNNING". Mentre il job è in questo stato, potrebbe essere necessario accedere ad alcuni file che verranno copiati nel WN;
- Fase 6: se il job è stato eseguito correttamente e senza errori, l'output prodotto durante l'esecuzione nel WN verrà salvato nell'SE aggiornando l'LFC oppure verrà copiato in locale nel percorso precedentemente indicato dall'utente. L'LB cambia lo stato del job in "DONE";
- Fase 7: l'utente può recuperare l'output del job. Lo stato del job diventa "CLEARED".

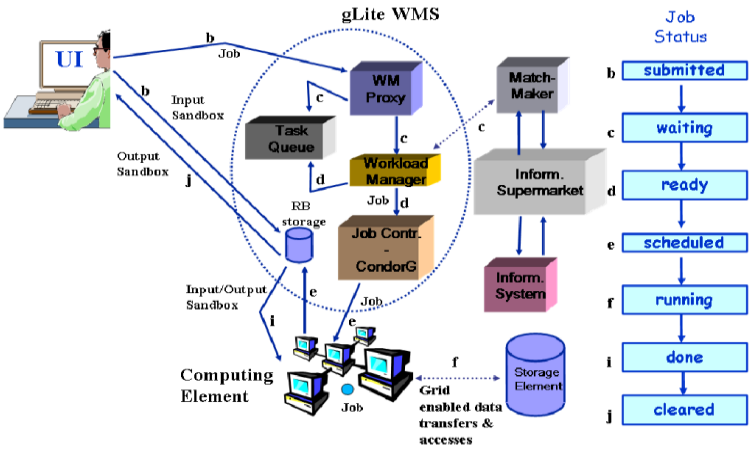

Figura 7 Fasi di sottomissione di un job

Le fasi di sottomissione di un job possono essere riassunte in questo modo:

- Submitted: il job è stato creato sulla UI ma è non ancora sottomesso per l'esecuzione;
- Waiting: durante questa fase il job è processato da parte dell'RB;
- Ready: il job è stato processato ma non inviato al CE di destinazione;
- Scheduled: il job è nel CE che decide quale WN deve eseguirlo;
- Running: Il job è eseguito in un WN;
- Done: il job ha terminato la sua esecuzione;
- Aborted: il job è stato abortito dal WMS;
- Cancelled: il job è stato annullato dall'utente;
- Cleared: il job ha terminato e l'output è stato scaricato.

3.2.9 Sun Grid Engine come batch system open source

Come descritto nel capitolo 2, attualmente nel cluster vengono usati due RMS:

- SGE (Sun Grid Engine): progetto software inizialmente supportato da SUN Microsystems, attualmente acquisita da Oracle, disponibile sia in versione a pagamento che open source. La versione open source installata nel cluster è la 6.2u5;
- LSF (Load Sharing Facility): software proprietario distribuito dalla Platform. La versione installata nel cluster è la 6.2. LSF è uno dei job manager supportato da GRAM (Grid Resource Allocation Manager), un componente di Globus Toolkit.

Inizialmente, è stato scelto di testare il funzionamento di SGE sebbene in quel momento fosse un RMS non ancora totalmente supportato da gLite. In questo modo è stato possibile testare e studiare l'affidabilità e la produttività di un RMS open source e poterlo confrontare con uno di tipo commerciale come LSF già in uso e funzionante nel cluster di Monserrato. Dopo aver installato SGE in tutti i nodi del cluster, si è provveduto abbastanza facilmente alla sua integrazione in ambito HPC e in fase sperimentale sulla Grid configurandolo come batch system per il CE di sito con opportuna coda per la VO CyberSAR. In seguito sono stati eseguiti una serie di test di prova i cui risultati positivi hanno permesso la creazione di ulteriori code di sottomissione in modo da consentire anche a utenti provenienti da VO esterne a CyberSAR l'esecuzione dei loro job. L'affidabilità dimostrata in questa fase di produzione ha mostrato prestazioni abbastanza simili a quelle di LSF. Questa configurazione ha permesso di destinare tutte le risorse di calcolo presenti nel cluster all'esecuzione di job di tipo HPC e Grid. Successivamente metà dei nodi sono rimasti sotto la gestione SGE mentre l'altra metà è stata configurata per essere gestita con l'installazione di LSF presente sul sito di Monserrato. Tramite il file system GPFS, è stato possibile condividere l'utilizzo dello Storage Element, assente nel sito di ingegneria, e permettere l'esecuzione dei job relativi all'esperimento ALICE. Sarebbe stato possibile installare l'intera configurazione necessaria per la gestione e l'esecuzione di tali job, ma dato l'esiguo numero di macchine fisiche disponibili, piuttosto che replicare numerosi servizi e rinunciare a nodi di calcolo in favore di questi ultimi, si è preferito estendere e condividere le risorse del cluster di Monserrato. La necessità di poter disporre di ulteriori nodi di calcolo e di risorse in generale, è diventata col tempo ancora più stringente. Ciò ha motivato la decisione di ricorrere all'uso della virtualizzazione, descritta nel capitolo 4, come strumento in grado di concentrare in pochi server tutti i servizi locali necessari per il funzionamento della Grid e liberare di conseguenza le macchine fisiche precedentemente riservate a tali servizi.

3.2.10 Configurazione della Grid

Come descritto precedentemente, date le dimensioni ridotte del polo Grid di ingegneria ed il numero molto elevato di elementi di cui il sistema Grid ha bisogno, è stato deciso di installare solo le componenti strettamente necessarie al sito, cioè: una UI, un CE ed il resto dei nodi alla funzione di WN. Per tutti gli altri elementi, che sono comunque necessari per il funzionamento, ci si è appoggiati all'infrastruttura Grid di Monserrato. La scelta del middleware di Grid è stata molto vincolante dal punto di vista dell'installazione e della configurazione di tutto il sistema Grid. Inizialmente è stata installata la versione 3.1 di gLite, in quel momento la più recente disponibile, poi aggiornata alla versione 3.2. I requisiti del middleware imponevano la scelta del sistema operativo da utilizzare ad una particolare distribuzione Linux con kernel appositamente modificato: la Scientific Linux 4.6 derivata da Red-Hat. Ciò ha creato non pochi problemi di gestione, manutenzione e aggiornamento di

tutti gli altri servizi e software installati oltre al middleware. Oltre a ciò, l'insieme dei repository dai cui recuperare tutti i pacchetti relativi a Grid, è stato spesso instabile e inconsistente, rilevando dipendenze non soddisfatte o addirittura la mancanza di un determinato pacchetto. La documentazione stessa reperibile in rete, è risultata alquanto carente e poca chiara su diversi aspetti. Probabilmente queste problematiche sono dovute alla grossa complessità dell'architettura middleware ed al suo stato di continuo sviluppo.

Installazione e configurazione del CE: di seguito viene descritta la procedura utilizzata per l'installazione del CE. Dopo aver scaricato i file "repo" necessari o averne recuperato le informazioni, è necessario posizionarli all'interno della cartella "/etc/yum.repos.d/". Solitamente i repository necessari sono i seguenti: dag.repo, jpackage.repo, ig.repo, lcg-ca.repo, glite-lcg_ce.repo e glite-bdii.repo. A questo punto è possibile lanciare l'installazione ripulendo la cache e aggiornando la lista pacchetti disponibili:

```
# yum clean all
# yum update
```

Prima di proseguire con l'installazione è necessario installare la jdk-1.5.0:

```
# yum install java-1.5.0-sun-compat -y
```

e poi procedere con l'installazione dei pacchetti relativi allo specifico profilo del CE: la lista delle Certification Authority, il CE lcg, il BDII di sito ed il pacchetto che permette al job manager che si intende utilizzare, nel nostro caso SGE, di dialogare con il CE.

```
# yum install lcg-CA
# yum install lcg-CE
# yum install glite-BDII
# yum install glite-SGE_utils
```

Terminata la fase di installazione si può procedere con quella di configurazione, effettuate tramite YAML (YAML Ain't Markup Language), installato di default nella cartella "/opt/glite/yaim/". In generale si tratta di impostare un file di variabili che sarà poi letto dalla procedura di configurazione. Come descritto in precedenza, è necessario possedere i certificati relativi all'host che ospiterà il CE richiedendoli alla CA di competenza. I file contengono la chiave pubblica "hostcert.pem" e la chiave privata "hostkey.pem" e devono essere posizionati nella cartella "/etc/grid-security" con i seguenti permessi:

```
# chmod 644 hostcert.pem
rw-r—r-- hostcert.pem
# chmod 600 hostkey.pem
rw------- hostkey.pem
```

Per procedere con la configurazione bisogna creare o editare i seguenti file di configurazione:

- ig_site_info.def: contiene le variabili di configurazione principale;
- ig_cyb_groups.conf: contiene le informazioni relative ai gruppi della Grid;
- ig_cyb_users.conf: contiene le informazioni relative agli utenti della Grid;
- wn_cyb_list.conf: contiene le informazioni relative ai WN utilizzati.

A questo punto si può lanciare il comando che effettua la configurazione del CE (parametro "-c"), usando le variabili definite nel file "siteinfo" (parametro "-s"), insieme ai profili previsti per il tipo di nodo (parametro "-n"):

```
# /opt/glite/yaim/bin/yaim -c -s ig_site_info.def -n lcg-CE -n SGE_utils –n
ig_BDII_site
```

L'output di questo comando è molto lungo ed è fondamentale una sua buona lettura per scoprire eventuali messaggi di errore.

Installazione e configurazione del WN: di seguito viene descritta la procedura utilizzata per l'installazione dei WN. Dopo aver scaricato i file "repo" necessari o averne recuperato le informazioni, è necessario posizionarli all'interno della cartella "/etc/yum.repos.d/". Solitamente i repository necessari sono i seguenti: dag.repo, ig.repo, jpackage.repo, lcg-ca.repo e glite-wn.repo. A questo punto è possibile lanciare l'installazione ripulendo la cache e aggiornando la lista pacchetti disponibili:

```
# yum clean all
# yum update
```

Prima di proseguire con l'installazione è necessario installare la jdk-1.5.0:

```
# yum install java-1.5.0-sun-compat -y
```

e poi procedere con l'installazione dei pacchetti relativi allo specifico profilo del WN: la lista delle Certification Authority ed il WN.

```
# yum install lcg-CA
# yum install glite-WN
```

In maniera simile a quanto visto per il CE, terminata la fase di installazione si può procedere con quella di configurazione, creando o editando i seguenti file di configurazione:

- ig_site_info.def: contiene le variabili di configurazione principale;
- ig_cyb_groups.conf: contiene le informazioni relative ai gruppi della Grid;
- ig_cyb_users.conf: contiene le informazioni relative agli utenti della Grid;
- wn_cyb_list.conf: contiene le informazioni relative ai WN utilizzati.

A questo punto si può lanciare il comando che effettua la configurazione del WN (parametro "-c"), usando le variabili definite nel file "siteinfo" (parametro "-s"), insieme al profilo previsto per il tipo di nodo (parametro "-n"):

```
# /opt/glite/yaim/bin/yaim -c -s ig_site_info.def -n glite-WN
```

Anche in questo caso l'output di questo comando è molto lungo ed è fondamentale una sua buona lettura per scoprire eventuali messaggi di errore.

Installazione e configurazione della UI: di seguito viene descritta la procedura utilizzata per l'installazione di una UI. Dopo aver scaricato i file "repo" necessari o averne recuperato le informazioni, è necessario posizionarli all'interno della cartella "/etc/yum.repos.d/". Solitamente i repository necessari sono i seguenti: dag.repo, jpackage.repo, lcg-ca.repo, glite-ui.repo e ig.repo. A questo punto è possibile lanciare l'installazione ripulendo la cache e aggiornando la lista pacchetti disponibili:

```
# yum clean all
# yum update
```

Prima di proseguire con l'installazione è necessario installare la jdk-1.5.0:

```
# yum install java-1.5.0-sun-compat -y
```

e poi procedere con l'installazione dei pacchetti relativi allo specifico profilo della UI: la lista delle Certification Authority e la UI.

```
# yum install lcg-CA
# yum install ig_UI_noafs
```

In maniera simile a quanto visto per il CE ed il WN, terminata la fase di installazione si può procedere con quella di configurazione, creando o editando i seguenti file di configurazione:

- ig_site_info.def: contiene le variabili di configurazione principale;
- ig_cyb_groups.conf: contiene le informazioni relative ai gruppi della Grid;
- ig_cyb_users.conf: contiene le informazioni relative agli utenti della Grid;
- wn_cyb_list.conf: contiene le informazioni relative ai WN utilizzati.

A questo punto si può lanciare il comando che effettua la configurazione della UI (parametro "-c"), usando le variabili definite nel file "siteinfo" (parametro "-s"), insieme al profilo previsto per il tipo di nodo (parametro "-n"):

```
# /opt/glite/yaim/bin/yaim -c -s /root/site_info/ig_site_info.def -n ig_UI_noafs
```

Anche in questo caso l'output di questo comando è molto lungo ed è fondamentale una sua buona lettura per scoprire eventuali messaggi di errore.

Per verificare il corretto funzionamento del CE e del WN, dalla UI dovrebbe essere possibile visualizzare i CE e le risorse disponibili con il seguente comando:

```
# lcg-info --list-ce --vo gilda
CE: nodo01.grid.vm:2119/jobmanager-lcglsf-cert
CE: nodo01.grid.vm:2119/jobmanager-lcglsf-infinite
CE: nodo01.grid.vm:2119/jobmanager-lcglsf-long
CE: nodo01.grid.vm:2119/jobmanager-lcglsf-short
```

ed inoltre, effettuare un test di esecuzione utilizzando una coda di sottomissione in uno dei CE disponibili:

```
# globus-job-run nodo1.grid.box/jobmager-lcglsf -q short /bin/hostname
```

Il file ".def": il file di configurazione ig_site_info.def, contiene tutte le variabili necessarie per la configurazione dei profili che si intende installare. Piuttosto che avere un unico file con tutte le variabili è anche possibile suddividerle in base al profilo di appartenenza in modo da semplificarne l'utilizzo. Di seguito viene data una descrizione di quelle più rilevanti:

Variabili relative ai percorsi dei file utilizzati durante la configurazione:

- USERS_CONF: percorso del file che contiene le informazioni relative ai utenti;
- GROUPS_CONF: percorso del file che contiene le informazioni relative ai gruppi;
- JAVA_LOCATION: percorso dei file relativi a java.

Variabili relative alle impostazioni della configurazione di rete:

- NTP_HOST_IP: contiene l'indirizzo IP del server NTP per la sincronizzazione temporale del sito;
- My_INT_DOMAIN: contiene il nome del dominio a cui il sito appartiene;
- INT_NET: contiene l'indirizzo IP privato della rete.

Variabili relative al CE: esistono una serie di variabili che non riporteremo che contengono le informazioni relative alle caratteristiche tecniche del CE, ovvero al modello, all'architettura e

al processore. Oltre a queste, sono presenti una serie di variabili, che contengono le informazioni relative gli indirizzi di altri elementi della Grid con cui il CE deve comunicare:

- WMS_HOST: contiene l'hostname corrispondente al servizio di WMS;
- LB_HOST: contiene l'hostname corrispondente al servizio di LB;
- PX_HOST: contiene l'hostname corrispondente al servizio di MyProxy;
- GRID_TRUSTED_BROKERS: contiene la lista dei DN (Distinguished Name) dei certificati degli host che sono ritenuti affidabili dal Proxy. E' necessario specificare quelli relativi al Resource brokers, WMS e FTS servers;
- DGAS_HLR_RESOURCE: contiene l'hostname corrispondente al servizio HLR (Home Location Register) del sito, usato per effettuare monitoring delle risorse.

Variabili relative al LFC (LCG Catalogue). Questo servizio memorizza le informazioni locali di ciascun sito:

- LFC_HOST: contiene l'hostname corrispondente al servizio LFC;
- LFC_DB_PASSWORD: contiene la password al database che contiene le informazioni del LCG File Catalogue;

Variabili di configurazione del Batch Server. Queste variabili consentono di configurare i percorsi relativi allo scheduler utilizzato dal CE:

- BATCH_SERVER: solitamente corrisponde al CE;
- JOB_MANAGER: contiene il nome del job manager utilizzato nel sito;
- CE_BATCH_SYS: contiene il nome del job manager utilizzato nel sito;
- BATCH_BIN_DIR: contiene il percorso dell'eseguibile relativo al job manager utilizzato;
- BATCH_VERSION: contiene la versione del job manager utilizzato;
- BATCH_LOG_DIR: contiene il percorso dove devono essere notificati i file di log relativi al job manager;
- SGE_ROOT: contiene il percorso in cui è stato installato job manager.

Variabili di configurazione del BDII:

- BDII_HOST: contiene l'hostname corrispondente al servizio BDII;
- SITE_BDII_HOST: contiene l'hostname corrispondente al servizio BDII di sito;
- BDII_REGIONS: contiene l'elenco dei servizi che sono supportati nel sito come CE e SE;
- BDII_HTTP_URL: contiene l'URL del file di configurazione del BDII;

Configurazione delle variabili relative alle VO. Con queste variabili vengono dichiarate quali VO il nostro sito supporta e quali code vengono usate per esse:

- VOS: contiene le VO che vengono utilizzate nel sito;
- ALL_VOMS: contiene la lista di tutte le VO che sono supportate nel sito;
- QUEUES: contiene i nomi delle code che verranno utilizzate nel nostro sito;

Accesso alle risorse: come descritto in precedenza, per poter utilizzare l'infrastruttura Grid è necessario avere un account in una delle User Interface esistenti nel sito, essere iscritto ad una VO e possedere un certificato personale valido. L'utente dopo aver soddisfatto tali requisiti, deve copiare il proprio certificato personale nella UI in cui è autenticato:

```
$ mkdir ~/.globus
$ cp your_certificate.p12 ~/.globus
```

convertire il certificato in formato "pem" ed estrarne la chiave privata:

```
$ openssl pkcs12 -nocerts -in .globus/<your>.p12 -out userkey.pem
$ openssl pkcs12 -clcerts -nokeys -in .globus/<your>.p12 -out usercert.pem
```

impostare i seguenti permessi ai file:

```
$ chmod 644 ~/.globus/usercert.pem
$ chmod 400 ~/.globus/userkey.pem
```

creare il proxy che avrà una durata di default di 12 ore:

```
$ voms-proxy-init --voms vo_name
```

visualizzare le informazioni sul proxy creato:

```
$ voms-proxy-info –all
```

Da questo momento si è in grado di interagire con l'infrastruttura Grid e sarà possibile visualizzare le risorse disponibili con i seguenti comandi:

```
$ lcg-infosites --vo <your vo> ce
$ lcg-infosites --vo <your vo> se
```

I job possono essere sottomessi tramite l'uso del linguaggio JDL, dopo aver editato un file con tale estensione. All'interno di esso viene specificato il programma da eseguire, i file di input e output ed eventuali argomenti. Il file ".jdl" è composto da una serie di coppie attributo/valore. Di seguito vengono descritti i principali attributi che possono essere usati:

- Type: indica il tipo di richiesta, valori permessi sono Job, DAG, Collection;
- jobType: indica il tipo di Job, può essere usato quando il tipo di richiesta è Job. Altri valori sono: Normal, Interactive, MPICH, Parametric;
- Executable: indica il percorso in cui si trova l'eseguibile da utilizzare durante la sottomissione;
- Arguments: indica gli argomenti da riga di comando che devono essere passati all'eseguibile;
- StdOutput & StdError: indica i file nei quali dovrà essere salvato lo standard output e lo standard error prodotto dall'esecuzione del job;
- OutputSandbox: è il contenitore dei file prodotti dall'esecuzione del job;

Requirements & Rank: indicano le richieste in termini di risorse necessarie per l'esecuzione del job;

Esempio di file "jdl":

```
[
Type = "Job";
JobType = "Normal";
Executable = "/bin/hostname";
Arguments = "-f";
StdOutput = "std.out";
StdError = "std.err";
OutputSandbox = {"std.out","std.err"};
```

```
Requirements = other.GlueCEStateStatus == "Production";
Rank = ( -other.GlueCEStateEstimatedResponseTime );
]
```

Sottomissione di un job: i comandi da usare per sottomettere un job dipendono dal tipo di Resource Broker presente, nell'infrastruttura. Cybersar, nello specifico nel polo di Monserrato, ha due versioni di RB. La versione LCG (LHC Computing Grid) ha la seguente serie di comandi:

- **`edg-job-list-match job.jdl`**: visualizza le risorse disponibili che soddisfano i requisiti di sottomissione del job;
- **`edg-job-submit job.jdl`**: sottomette il job passato come parametro per l'esecuzione e restituisce un ID identificativo;
- **`edg-job-status job_id`**: monitora lo status del job e ne stampa lo stato. Durante la sua esecuzione, vengono attraversati una serie di stati che porteranno ad una terminazione con successo o ad un fallimento;
- **`edg-job-get-output --dir job_id`**: in caso di job terminato con successo, recupera l'output, temporaneamente archiviato nel Resource Broker, nella modalità specificata dall'attributo OutputSandbox. Il parametro "--dir" indica il percorso locale in cui memorizzare l'output.

La versione gLite ha la seguente serie di comandi:

- **`glite-wms-job-list-match -a job.jdl`**: visualizza le risorse disponibili che soddisfano i requisiti di sottomissione del job. L'opzione "-a" esegue la delegazione automatica del proxy ad ogni sottomissione;
- **`glite-wms-job-submit -a job.jdl`**: sottomette il job passato come parametro per l'esecuzione e restituisce un ID identificativo;
- **`glite-wms-job-status job_id`**: monitora lo status del job e ne stampa lo stato. Durante la sua esecuzione, vengono attraversati una serie di stati che porteranno ad una terminazione con successo o ad un fallimento;
- **`glite-wms-job-output --dir job_id`**: in caso di job terminato con successo, recupera l'output, temporaneamente archiviato nel Resource Broker, nella modalità specificata dall'attributo OutputSandbox. Il parametro "--dir" indica il percorso locale in cui memorizzare l'output.

4. VIRTUALIZZAZIONE E KERNEL VIRTUAL MACHINE

La prima definizione di macchina virtuale venne fornita in un articolo di Gerald J. Popek e Robert P. Goldberg del 1974 attraverso l'introduzione di un Virtual Machine Monitor (VMM). Il VMM viene definito come un software in esecuzione sulla macchina reale che ha il completo controllo delle risorse hardware da essa fornite. Esso crea degli ambienti di esecuzione che forniscono agli utenti l'illusione di un accesso diretto alle risorse della macchina fisica. Tali ambienti vengono detti Macchine Virtuali (VM). La definizione formulata da Popek e Goldberg è la seguente: "*Una macchina virtuale è un duplicato software di un computer reale nel quale un sottoinsieme statisticamente dominante di istruzioni del processore virtuale viene eseguito nativamente sul processore fisico*". Insieme a questa definizione vengono specificate anche le tre caratteristiche principali che il VMM deve possedere:

1. **Equivalenza:** garantire un ambiente di esecuzione per i programmi sostanzialmente identico a quello della macchina reale. Le uniche differenze sono legate all'overhead delle VMM ed alla ridotta disponibilità delle risorse;
2. **Efficienza:** garantire una elevata efficienza nell'esecuzione dei programmi. Quando possibile, il VMM deve permettere l'esecuzione diretta delle istruzioni impartite dalle macchine virtuali: le istruzioni non privilegiate vengono eseguite direttamente in hardware senza coinvolgere il VMM;
3. **Controllo delle risorse:** garantire la stabilità e la sicurezza dell'intero sistema. Il VMM deve rimanere sempre nel pieno controllo delle risorse hardware: i programmi in esecuzione nelle macchine virtuali come le applicazioni ed il sistema operativo, non possono accedere all'hardware in modo privilegiato.

In generale si può affermare che in un sistema caratterizzato da un insieme di risorse hardware e software, virtualizzare significa creare dei sostituti per le risorse reali presentando all'utente una visione di queste ultime, diversa da quella reale. Questi sostituti, chiamati risorse virtuali, hanno la stessa funzione e le stesse interfacce esterne delle loro controparti ma si differenziano per alcune caratteristiche e funzioni quali dimensioni, prestazioni e costi. La virtualizzazione di un sistema di elaborazione permette di trattare le risorse fisiche da un punto di vista logico, introducendo un livello di indirezione tra la vista logica virtuale e quella fisica reale in modo da disaccoppiare il comportamento delle risorse hardware e software dalla loro realizzazione fisica. Il disaccoppiamento è realizzato da un componente chiamato Virtual Machine Monitor (VMM, o Hypervisor) il cui compito è consentire l'utilizzo delle risorse di una singola piattaforma hardware da parte di più macchine virtuali (VM). In questo modo, tali risorse possono essere condivise da più sistemi operativi, ognuno dei quali è installato su una diversa macchina virtuale, costituita oltre che dall'applicazione che in essa viene eseguita, anche dal sistema operativo

utilizzato, vedi fig. 1.

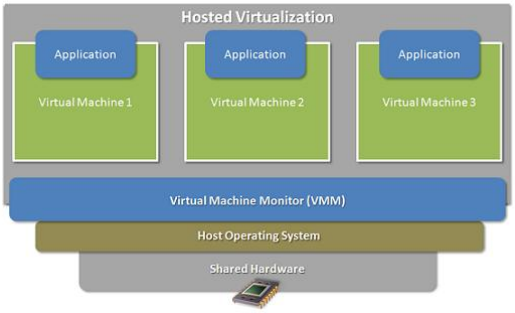

Figura 1 Schema di virtualizzazione

Il VMM è l'unico mediatore nelle interazioni tra le macchine virtuali e l'hardware sottostante. Le singole VM credono di avere a disposizione tutto l'hardware disponibile nella macchina fisica ma in realtà è l'hypervisor che fornisce questa illusione garantendo l'isolamento tra le VM e la stabilità del sistema.

4.1 Tecnologie di virtualizzazione

La virtualizzazione[11][25] può essere realizzata in diversi modi, ma le tecniche più usate sono la virtualizzazione completa, la paravirtualizzazione e l'emulazione. Nella realizzazione di un VMM, di fondamentale importanza sono il livello di esecuzione all'interno del sistema di elaborazione e la modalità di dialogo per l'accesso alle risorse fisiche tra macchina virtuale e VMM.

VMM di sistema: viene eseguito direttamente sopra l'hardware dell'elaboratore integrandone le funzionalità di virtualizzazione e permettendo alle macchine virtuali di usare la stessa interfaccia o istruzioni macchina dell'architettura fisica, vedi fig. 2. Questo tipo di approccio viene chiamato "Full virtualization". Hypervisor di questo tipo sono Linux con Kvm e Vmware ESXi. E' necessario che il VMM possieda tutti i driver necessari per pilotare le periferiche. In questo caso con il termine "Host" si indica la piattaforma di base sulla quale si realizzano le macchine virtuali denotate con il termine "Guest". Un approccio simile a questo, chiamato "Paravirtualization", è quello in cui il VMM presenta un'interfaccia diversa da quella dell'architettura hardware. In questo caso l'hypervisor opera in maniera più cooperativa poiché il sistema operativo ospite, consapevole di essere virtualizzato, collabora con esso per virtualizzare l'hardware sottostante. Esempi di questo tipo sono Linux con Xen e User Mode Linux (UML).

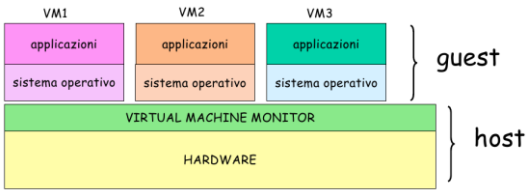

Figura 2 VMM di sistema

VMM ospitato: eseguito come applicazione sopra un sistema operativo esistente, opera nello spazio utente e accede l'hardware tramite system call del sistema operativo su cui viene installato e su cui può far riferimento per la gestione delle periferiche e l'utilizzo di servizi come ad esempio lo scheduling, vedi fig. 3. In questo caso la sua installazione risulta molto semplice, come una normale applicazione, ma le prestazioni sono inferiori

rispetto al modello precedente. Un esempio di VMM di questo tipo sono Virtualbox, Vmware player, Parallels, VirtualPCe Qemu;

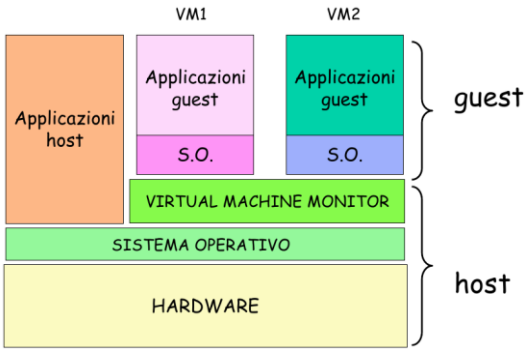

Figura 3 VMM ospitato

Emulazione: l'emulazione consiste nella capacità di eseguire programmi compilati per un certo insieme di istruzioni su un sistema di elaborazione dotato di un diverso insieme di istruzioni. Le singole istruzioni dell'architettura ospitata, infatti, vengono emulate interamente permettendo anche a sistemi operativi con architettura differente di poter essere eseguiti senza modifiche. Se da un lato l'emulazione offre il vantaggio dell'interoperabilità tra ambienti eterogenei, dall'altro esistono problemi di efficienza e ripercussioni sulle prestazioni. Questa tecnica si è a sua volta evoluta in interpretazione e ricompilazione dinamica:

- l'interpretazione si basa sulla lettura di ogni singola istruzione del codice macchina che deve essere eseguito e sulla esecuzione di più istruzioni sull'host virtualizzante per ottenere semanticamente lo stesso risultato. E' un metodo molto generale e potente che presenta una grande flessibilità nell'esecuzione perché consente di emulare e riorganizzare i meccanismi propri delle varie architetture. Vengono, ad esempio, normalmente utilizzate parti di memoria per salvare il contenuto dei registri della CPU emulata, registri che potrebbero non essere presenti nella CPU emulante. Produce un sovraccarico mediamente molto elevato poiché possono essere necessarie molte istruzioni dell'host per interpretare una singola istruzione sorgente;
- la ricompilazione dinamica piuttosto che leggere una singola istruzione del sistema ospitato come l'interpretazione, legge interi blocchi di codice, li analizza, li traduce per la nuova architettura ottimizzandoli e infine li mette in esecuzione con un vantaggio in termini di prestazioni evidente. Il codice ricompilato infatti viene tradotto e ottimizzato, utilizzando tutte le possibilità offerte dalla nuova architettura e messo in esecuzione. Parti di codice utilizzate frequentemente possono essere bufferizzate per evitare di doverle ricompilare in seguito. La maggior parte dei più noti emulatori come QEMU e Bochs, utilizzano questa tecnica per attuare la virtualizzazione.

4.1.1 Vantaggi della virtualizzazione

I vantaggi della virtualizzazione sono numerosi e tali che i maggiori produttori di processori hanno incluso nelle loro cpu il supporto alla virtualizzazione in maniera diretta. Qui di seguito vengono descritti alcuni dei benefici apportati da questa tecnica:

- l'uso di più sistemi operativi sulla stessa macchina fisica permette più ambienti di esecuzione eterogenei per lo stesso utente e la possibilità di esecuzione di applicazioni concepite per un particolare sistema operativo.
- l'isolamento degli ambienti di esecuzione di ogni macchina virtuale, detto sandbox, offre la possibilità di effettuare testing di applicazioni preservando l'integrità degli altri ambienti e del VMM. Inoltre, eventuali attacchi da parte di malware o spyware sono confinati alla singola macchina virtuale;
- la possibilità di concentrare più macchine su un'unica architettura hardware, facendo una operazione di consolidamento delle risorse, ne permette un utilizzo più efficiente ed un abbattimento dei costi di amministrazione e di gestione. Alcuni studi effettuati in grossi centri di elaborazione dati, hanno dimostrato che solo una piccola parte delle risorse totali di un server è effettivamente utilizzato.
- la possibilità di assegnare diverse macchine virtuali a più utenti, permette in ambito didattico, di realizzare laboratori in cui ogni studente può usare una macchina virtuale personale da amministrare autonomamente con la possibilità di esercitarsi senza limitazioni nelle tecniche di amministrazione e configurazione del sistema, installare e provare nuovi sistemi operativi e applicazioni anche potenzialmente pericolose senza il rischio di compromettere la funzionalità dell'intero sistema.

4.2 Linux e Kvm

Il Kernel-based Virtual Machine[12][13] è una tecnica di virtualizzazione completa per Linux su hardware x86 con estensione alla virtualizzazione “Intel VT” o “AMD-V”. La tecnica usata da Kvm è quella di trasformare un Kernel Linux in un hypervisor nativo semplicemente tramite il caricamento di un modulo. Questa operazione, esporta un dispositivo a caratteri chiamato “/dev/kvm”, il quale abilita una terza modalità detta “guest” che va ad aggiungersi a quelle tradizionali. Un normale processo Linux, infatti, ha due modalità di esecuzione: la “user mode”, utilizzata di default per le applicazioni e la “kernel mode” utilizzata nel momento in cui l'applicazione richiede qualche servizio dal Kernel come ad esempio quando deve scrivere sull'hard disk. La modalità “guest mode” di Kvm contiene al suo interno le modalità standard di Linux. Tale modalità, viene utilizzata da tutti i processi eseguiti all'interno di una macchina virtuale tranne quelli relativi all'I/O poiché gestiti in maniera indipendente. Le macchine virtuali utilizzano il nuovo dispositivo per avere uno spazio di indirizzamento riservato, separato da quello del Kernel e da qualsiasi altra VM in esecuzione. Il fatto di poter usare il Kernel di Linux come hypervisor, permette di poter eseguire macchine virtuali come se fossero dei normali processi schedulati dallo scheduler di Linux, in grado di ospitare sistemi operativi separati ed utilizzare tutte le funzionalità ed i miglioramenti offerti dallo sviluppo del Kernel stesso, vedi fig. 4.

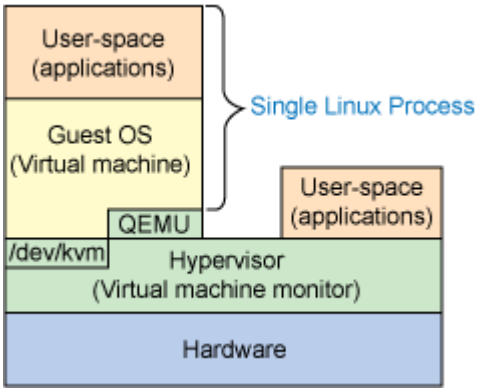

Figura 4 Kvm nel Kernel Linux

Kvm gestisce le operazioni di I/O principalmente tramite QEMU, una soluzione software che permette di virtualizzare l'intero ambiente di un PC inclusi hard disk, schede grafiche e dispositivi di rete. Qualsiasi richiesta di I/O, infatti, viene intercettata e gestita tramite la modalità "user mode" del sistema operativo host in modo da essere emulata dal processo QEMU. Come descritto in precedenza, ogni sistema operativo guest ha il suo spazio di indirizzi di memoria, mappati quando la VM viene istanziata e gestiti tramite un insieme di tabelle di "shadow page" che permettono la traduzione degli indirizzi fisici della macchina host in indirizzi fisici del sistema guest. Una funzionalità interessante è il Kernel Same-page Merging (KSM), che permette la condivisione di una stessa pagina di memoria tra VM diverse eseguendo il merging di due pagine di memoria identiche. Per memorizzare le immagini delle macchine virtuali è possibile utilizzare dischi locali, dispositivi SCSI o dispositivi collegati via rete permettendone la condivisione tra hosts diversi. Kvm permette inoltre, di eseguire operazioni come la "live migration" che consente di spostare una macchina virtuale in esecuzione senza interruzione del servizio oppure poterne salvare lo stato corrente per una esecuzione successiva. Per quanto riguarda la sicurezza, poiché le VM sono dei normali processi in esecuzione, è possibile applicare tutti gli strumenti disponibili in Linux per isolare, restringere e monitorare le risorse da queste utilizzate. I sistemi operativi ospiti "guest" supportati sono oltre le distribuzioni Linux: Microsoft Windows, MS DOS, OpenBSD, FreeBSD, OpenSolaris e Solaris x86.

4.3 Qemu

Qemu[14] è un emulatore open source che supporta due modalità operative: "user mode" e "system mode". L'emulazione "user mode", grazie all'uso della ricompilazione dinamica delle istruzioni, permette ad un processo creato per una determinata cpu di essere eseguito in un altra. La "system mode" permette invece l'emulazione di un intero sistema di elaborazione. Uno dei pregi più interessanti è la sua velocità di emulazione ottenuta grazie alla tecnica di traduzione dinamica delle istruzioni che permette l'esecuzione di programmi già compilati su architetture diverse. La traduzione dinamica, oltre ad essere efficiente e portabile, avviene in due fasi principali: la prima converte le istruzioni "target" in micro-operazioni in codice C, la seconda crea una mappa tra le istruzioni e le relative operazioni. Inoltre, durante questo processo, vengono memorizzati blocchi di 16 Mbyte di codice per minimizzare l'overhead della traduzione. QEMU, oltre ad una vasta varietà di periferiche, è in grado di emulare sistemi x86, AMD64, Power PC, MIPS e ARM.

4.4 Libvirt

Libvirt è una libreria che fornisce un insieme di API standard, che consentono una maggiore astrazione rispetto all'hypervisor in uso, per la creazione e gestione di un sistema operativo guest chiamato “domain”, in esecuzione su di una macchina fisica chiamata “node”. In libvirt, ci sono due distinte modalità di controllo. Nella prima, l'applicazione di gestione ed i domini coesistono nello stesso nodo, in questo caso l'applicazione di gestione lavora attraverso libvirt per controllare i “domains” locali, vedi fig. 5.

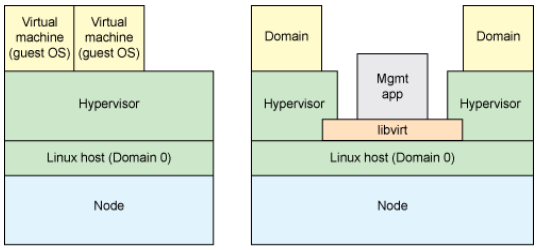

Figura 5 Applicazione di gestione e domini nello stesso nodo

Nella seconda, l'applicazione di gestione ed i domini stanno su nodi separati per cui è necessaria una comunicazione remota che fa uso di un particolare demone chiamato “libvirtd”, vedi fig. 6.

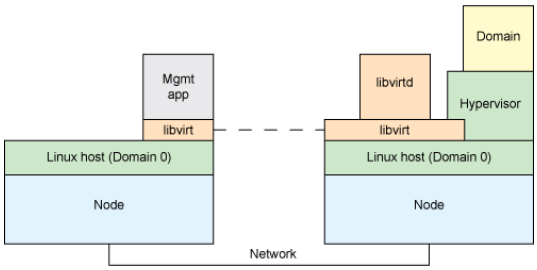

Figura 6 Applicazione di gestione e domini su nodi diversi

Questo demone, fatto partire automaticamente durante l'installazione della libreria libvirt, determina l'hypervisor locale e configura i relativi driver grazie ad una architettura driver-based, rendendo possibile l'esecuzione delle funzionalità implementate negli hypervisors supportati, tra cui Kvm, Xen, Qemu, Virtualbox ed altri. Per la gestione ed il monitoraggio di sistemi in cui si utilizza questa libreria in combinazione con Kvm, possono essere adoperati i seguenti tool:

- Virt-install: interfaccia a riga di comando che può essere usata in maniera interattiva o tramite script in modo da automatizzare la creazione delle VMs;
- Virtual Machine Manager (virt-manager): permette la creazione, la gestione ed il monitoraggio di VMs dal proprio desktop tramite interfaccia grafica;
- Virsh (virtualization shell): interfaccia a riga di comando che permette la creazione, la gestione ed il monitoraggio di VMs usando singolarmente i comandi disponibili oppure eseguendoli in modalità interattiva tramite la propria shell. Le VMs sono create in base ad un file XML descrittivo. La maggior parte dei comandi, richiedono i privilegi di amministratore tranne alcuni che permettono operazioni di sola lettura. Per il suo funzionamento, è necessario installare e mandare in esecuzione la libreria “libvirt” di cui Virsh usa le API.

Di seguito viene mostrato un esempio di come lavora la libreria libvirt con

l'applicazione “Virsh” costruita al di sopra di essa. Per prima cosa bisogna definire il file xml di configurazione del dominio nel quale sono specificate tutte le opzioni necessarie per il suo funzionamento:

```
<xml version="1.0"?>
<domain type='qemu'>
  <name>Domain_Name<name>
  <uuid><uuid>
  <memory>131072<memory>
  <currentMemory>131072<currentMemory>
  <vcpu>1<vcpu>
  <os>
    <type arch='x86_64' machine='pc'>hvm<type>
  <os>
  <devices>
    <emulator>/usr/bin/qemu<emulator>
    <disk type='file' device='disk'>
      <source file='/home/image.img'/>
      <target dev='hda'/>
    <disk>
    <interface type='network'>
      <source network='default'/>
    <interface>
    <graphics type='vnc' port='-1'/>
  <devices>
<domain>
```

In questo semplice esempio sono definiti: il tipo ed il nome del dominio a cui libvirt assegnerà un UUID (Universally Unique Identifier), la quantità massima di memoria, il numero di processori virtuali, il tipo di macchina da emulare, il disco virtuale per il dominio, la configurazione di rete e l'uso di VNC (Virtual Network Computing) per monitoraggio della VM. Per mandare in esecuzione un dominio:

```
$ virsh create domain.xml
Connecting to uri: qemu:///system
Domain Domain_Name created from domain.xml
```

In questo caso la URI usata per connettersi al dominio è quella relativa al driver locale di Qemu, nel caso si volesse connettere un dominio remoto tramite “ssh” potremmo usare la seguente URI: “qemu+ssh://host/”. E' possibile listare tutti i domini attivi con il comando:

```
$ virsh list
Connecting to uri: qemu:///system
 Id Name                 State
----------------------------------
  1 Domain_Name      running
```

Virsh mette a disposizione numerosi comandi di gestione che permettono il salvataggio, la sospensione, il ripristino ed il riavvio di un dominio. E' inoltre possibile creare un file di configurazione da un dominio in esecuzione con il comando:

```
$ virsh dumpxml [domain-id, domain-name or domain-uuid]
```

4.5 Consolidamento dei servizi Grid

Lo studio e la prova di alcune delle tecniche di virtualizzazione descritte precedentemente, hanno consentito di individuare quella più idonea per l’utilizzo più efficiente delle risorse disponibili. Poiché nell’hardware a disposizione è presente il supporto alla virtualizzazione, la scelta di utilizzare il sistema operativo Linux con Kvm

come hypervisor, si è rivelata quella più breve in termini di tempo e di più facile implementazione e configurazione. Inoltre, l'utilizzo della libreria libvirt e del tool virt-manager ha consentito una facile gestione e monitoraggio delle macchine virtuali. I test di funzionamento sono stati eseguiti in prima istanza su macchine virtuali la cui immagine era memorizzata nell'hard disk della macchina fisica su di un file in modo da poterle modificare ed eventualmente ricreare in maniera agevole e poco vincolante. Considerando però le scarse prestazioni ottenute soprattutto durante le operazioni di I/O, in una fase successiva ed a configurazione pressoché completata, le immagini sono state trasferite su LVM (Logical Volume Manager) in modo da ottenere prestazioni più performanti. La necessità di dover disporre di ulteriori nodi di calcolo ha evidenziato il fatto che alcuni dei servizi locali utilizzati per il funzionamento della Grid, come il Computing Element, i Worker Nodes e lo stesso job manager SGE, disponevano inizialmente di una intera macchina fisica per ciascuno con conseguente uso inefficiente delle risorse. La soluzione è stata quella di installare e configurare nuovamente tali servizi dedicando loro una macchina virtuale consentendo un uso più razionale delle risorse disponibili e permettendo di ottenere ulteriori benefici come il contenimento delle alte temperature nei periodi di minore sottomissione dei job e la riduzione generale del consumo di energia elettrica. La sperimentazione effettuata sulle tecniche di virtualizzazione ha permesso, inoltre, la realizzazione di un sistema di Cloud computing, come descritto successivamente nel capitolo 5.

4.6 Istanziare un sistema operativo guest

Di seguito viene mostrato un esempio di esecuzione di una istanza di un nuovo sistema operativo guest, utilizzando una cpu con il supporto alla virtualizzazione. La verifica può essere effettuata in questo modo:

```
$ grep -E 'vmx|svm' /proc/cpuinfo
```

Quando si esegue il comando "kvm", viene invocato il relativo modulo kvm che interagendo con il dispositivo "/dev/kvm", tramite un insieme di "ioctls" crea un nuovo oggetto di tipo VM a cui verrà associata una cpu ed un disco virtuale per poi eseguirne il boot. E' quindi necessario creare un disco immagine per il sistema operativo ospite:

```
$ qemu-img create -f raw vm-disk.img 5G
```

oppure:

```
$ qemu-img create -f qcow vm-disk.img 5G
```

L'uso del formato "qcow", del quale si tratterà nel capitolo 5, permette di creare immagini la cui dimensione è pari allo spazio reale inizialmente occupato dal file su disco. Tale spazio tenderà ad aumentare durante l'utilizzo della macchina virtuale. A questo punto possiamo installare all'interno del nuovo disco il sistema operativo guest assumendo di averlo disponibile in una ISO su cd-rom e di fare il boot da questo supporto:

```
$ kvm -no-acpi -m 384 -cdrom guestos.iso -hda vm-disk.img -boot d
```

Ad installazione terminata sarà possibile eseguire il boot dell'immagine:

```
$ kvm vdisk.img -m 384
```

5. CLOUD COMPUTING

Il Cloud computing, anche se fortemente legato al Grid computing come modello di calcolo distribuito pensato per essere utilizzabile in maniera simile alle tradizionali utenze come acqua, luce, gas e telefonia, si differenzia da esso per un approccio nettamente differente. Il termine "Cloud" denota l'infrastruttura come una "nuvola" da cui le imprese e gli utenti possono accedere alle loro applicazioni come se stessero richiedendo un servizio. Negli ultimi anni è diventato sempre più presente nel panorama informatico con un numero sempre più crescente di offerte commerciali. Tra le numerose soluzioni per la realizzazione di sistemi Cloud verranno di seguito analizzate in particolare due applicazioni open source: OpenNebula e Eucalyptus.

5.1 Architettura, tipologie e componenti

Secondo il NIST[19], *"Il cloud computing è un modello abilitante un accesso comodo ed on-demand ad un pool condiviso di risorse di calcolo configurabili che possono essere velocemente ottenute e rilasciate con minimo sforzo di gestione ed una limitata interazione con il fornitore di servizi."*. Tale definizione è molto generale, e non vincolata ad alcuna specifica tecnologia o implementazione hardware e software. Per meglio caratterizzare il modello, il NIST si concentra su cinque caratteristiche essenziali che un sistema di Cloud ideale dovrebbe presentare:

- On-demand self-service: l'utilizzatore del servizio può richiedere ed utilizzare le risorse di calcolo e di storage secondo le sue necessità, senza richiedere nessuna interazione umana con il fornitore di servizi;
- Broad network access: le risorse di calcolo fornite dal servizio sono disponibili sulla rete e disponibili attraverso meccanismi standard che permettono l'utilizzo ad applicazioni eseguite su piattaforme eterogenee.
- Resource pooling: le risorse del fornitore di servizio sono raggruppate allo scopo di servire gli utilizzatori attraverso un modello di erogazione multi-tenant. Le risorse fisiche e virtuali sono assegnate dinamicamente secondo le esigenze degli utilizzatori.
- Rapid elasticity: le risorse possono essere fornite in modo elastico, veloce e, a volte, automatico permettendo una veloce scalabilità verso l'alto e verso il basso.
- Measured service: i sistemi cloud controllano ed ottimizzano automaticamente l'uso delle risorse facendo leva sulla capacità di misura dell'uso delle tipologie di servizio. L'utilizzo delle risorse può essere monitorato e controllato in modo trasparente per il fornitore di servizio e l'utilizzatore.

Le risorse sono presentate agli utenti in base ad uno dei tre diversi modelli di fornitura o servizio, vedi fig. 1:

- Software as a Service (SaaS): il servizio fornito agli utenti è quello di implementare le applicazioni del fornitore eseguite sull'infrastruttura cloud. Le applicazioni sono accessibili da diversi dispositivi client attraverso interfaccia a riga di comando o tramite thin client, come un web browser. Gli utenti, al massimo, possono gestire la configurazione dell'applicazione limitata al loro uso;
- Platform as a Service (PaaS): il servizio fornito agli utenti è quello di implementare sull'infrastruttura cloud le proprie applicazioni, sviluppate usando linguaggi di programmazione, librerie e strumenti supportati dal fornitore. Gli utenti hanno il controllo sulle applicazioni sviluppate e la possibilità di configurare le impostazioni;
- Infrastructure as a Service (IaaS): il servizio prestato agli utenti è quello di fornire le risorse di calcolo fondamentali (processing, storage, networks ecc.) dove gli utenti sono in grado di distribuire ed eseguire arbitrariamente il software, come sistemi operativi e applicazioni. Gli utenti hanno il controllo sui sistemi operativi, lo storage e le applicazioni distribuite e il controllo limitato della configurazione di rete (ad esempio, firewalls o prenotazione dell'indirizzo IP).

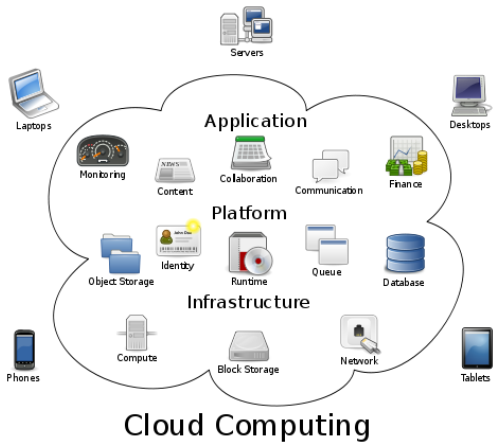

Figura 1 Diagramma logico di una rete Cloud computing

A seconda del modo in cui le risorse sono organizzate e rese disponibili agli utenti, possiamo distinguere tra quattro diversi modelli di distribuzione:

- Private cloud: l'infrastruttura cloud è fornita per l'uso esclusivo di una singola organizzazione;
- Community cloud: l'infrastruttura cloud è fornita per l'uso esclusivo di una specifica comunità di utenti proveniente da organizzazioni che ne condividono gli ambiti;
- Public cloud: l'infrastruttura cloud è fornita per un utilizzo aperto al pubblico. Essa può essere posseduta, gestita e amministrata da un'azienda, una università, o una organizzazione governativa, o una combinazione di queste.
- Hybrid cloud: l'infrastruttura cloud è una composizione di due o più infrastrutture cloud distinte (private, community, o public) che rimangono entità uniche, ma sono tenute insieme da una tecnologia standardizzata o proprietaria che permette la portabilità dei dati e delle applicazioni.

Mentre le infrastrutture di cloud private e community, possono essere possedute, gestite,

e controllate da organizzazioni, terze parti, o una combinazione di queste, e possono essere collocate o meno nei rispettivi locali, i public clouds sono collocati nei locali del fornitore di cloud. L'Open Cloud Manifesto[20], con il suo motto "dedicato a chi crede che il cloud dovrebbe essere aperto", pone l'apertura come uno dei principi fondamentali del cloud computing, integrando in tal modo il modello del NIST, immaginando una prospettiva di lungo termine basata su questo principio e insistendo sul fatto che proprio l'apertura favorisce il raggiungimento di un importante insieme di obiettivi:

- Choice: quando una organizzazione sceglie un fornitore, un'architettura o un modello di utilizzo, l'impiego di un cloud aperto renderà più semplice l'uso di un fornitore o di una architettura differente ogni volta che l'ambiente lavorativo cambia;
- Flexibility: non importa quale fornitore di cloud e architettura un organizzazione stia usando, un cloud aperto renderà più facile lavorare con altri gruppi, anche se questi abbiano scelto fornitori e architetture diverse. Un cloud aperto renderà più facile alle organizzazioni la collaborazione tra diversi fornitori di cloud.
- Speed and Agility: l'utilizzo di interfacce aperte consente alle organizzazioni di creare nuove soluzioni che integrano public clouds, e sistemi IT attuali. Poiché le condizioni delle organizzazioni cambiano, un cloud aperto permetterà all'organizzazione di rispondere con velocità e agilità.
- Skills: in presenza di svariati modelli di programmazione proprietari, è probabile che un professionista IT conosca solo alcuni di essi. Con un cloud aperto, ci sarà un insieme ridotto di nuove tecnologie da imparare (soprattutto quando vengono utilizzate le tecnologie esistenti), migliorando notevolmente le probabilità che l'organizzazione possa trovare qualcuno con le competenze necessarie.

5.2 OpenNebula 3.0

OpenNebula[16] è un toolkit open source di Cloud computing per la gestione di infrastrutture di elaborazione dati eterogenee e distribuite. Tra le diverse tipologie, è in grado di realizzare clouds privati, pubblici e ibridi in modalità Infrastructure as a Service. Le caratteristiche più rilevanti riguardano l'integrazione, la gestione, la scalabilità, la sicurezza ed il monitoraggio delle risorse oltre al supporto delle principali interfacce cloud quali Amazon EC2 e OGF OCCI. E' inoltre dotato di una architettura software molto flessibile che permette il suo utilizzo su infrastrutture con una grande varietà di hardware e software. Il progetto è sponsorizzato da C12G[18]. Le funzionalità principali sono, vedi fig. 2:

- Hosts and Virtualization: il virtualization manager supporta diversi hypervisors quali KVM, XEN e VMWARE, dando la possibilità di controllare e monitorare il ciclo di vita delle macchine virtuali;
- Storage and Images (Image Repository): il sottosistema di storage, utilizzato come repository per le immagini che verranno registrate in OpenNebula, è altamente personalizzabile e configurabile in modo da supportare filesystem condivisi o non. In OpenNebula una "immagine" o "image" è un file contenente un sistema operativo o dati, memorizzato nell'image repository, da cui possono essere istanziate più macchine virtuali. Una "macchina virtuale" o "virtual machine" è una istanza di un disco immagine registrato nel database di OpenNebula, in esecuzione

su di un cluster node;

- Networking: il sottosistema di gestione di rete, anch'esso altamente personalizzabile e configurabile, permette di integrare le funzionalità del toolkit con le diverse caratteristiche ed i requisiti dell'infrastruttura esistente;
- Interfaces & APIs: le interfacce sono utilizzate per la gestione delle risorse fisiche e virtuali. Oltre alla CLI (command line interface) e la GUI Sunstone, possono essere usati i comandi di esecuzione e gestione di macchine virtuali disponibili tramite web services in base agli standard OCCI (Open Cloud Computing Interface) e EC2 Query (Amazon EC2 Query Interface);
- Users and Groups: la creazione di utenti e di gruppi, insieme alle regole ACL (Access Control List) sono usati come meccanismo di autenticazione e autorizzazione di OpenNebula.

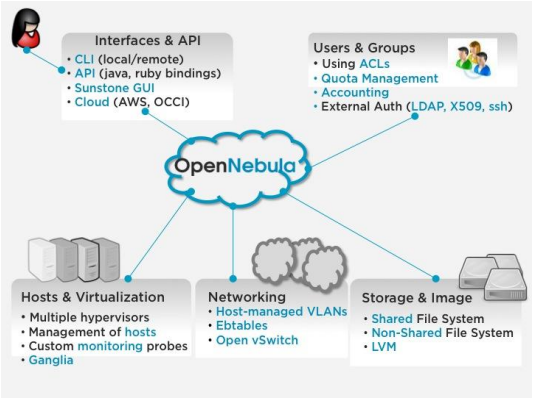

Figura 2 Schema delle funzionalità di OpenNebula

Le componenti principali di OpenNebula sono:

- Front-end: macchina fisica o virtuale nella quale vengono eseguiti i servizi di OpenNebula;
- Cluster Node: macchina fisica nella quale viene eseguito l'hypervisor supportato e l'istanza della macchina virtuale.

Il ciclo di vita delle macchina virtuali prevede diversi stati, qui vengono illustrati quelli principali, vedi fig. 3:

- Pending: non appena viene eseguito il comando di creazione è lo stato di default per una macchina virtuale e sta ad indicare che è in attesa di risorse. Rimarrà in tale stato fino a quando lo scheduler deciderà di eseguirne il deploy.
- Running: la macchina virtuale è in esecuzione, pronta all'uso e da questo momento in poi verrà periodicamente monitorata dal front end.
- Migrate**:** indica che la macchina virtuale sta migrando da una risorsa ad un altra. La migrazione può essere una live migration oppure cold migration.
- Stopped: indica che la macchina virtuale è stata fermata in seguito ad un comando di stop. Lo stato dell'istanza viene salvato trasferendone tutti i file dal cluster node al front-end.
- Suspended: simile allo stato stopped, ma i file vengono lasciati nel cluster node. Al suo riavvio non sarà necessario rischedulare la macchian virtuale.

- Failed: si è verificato un problema che non permette il passaggio allo stato running.

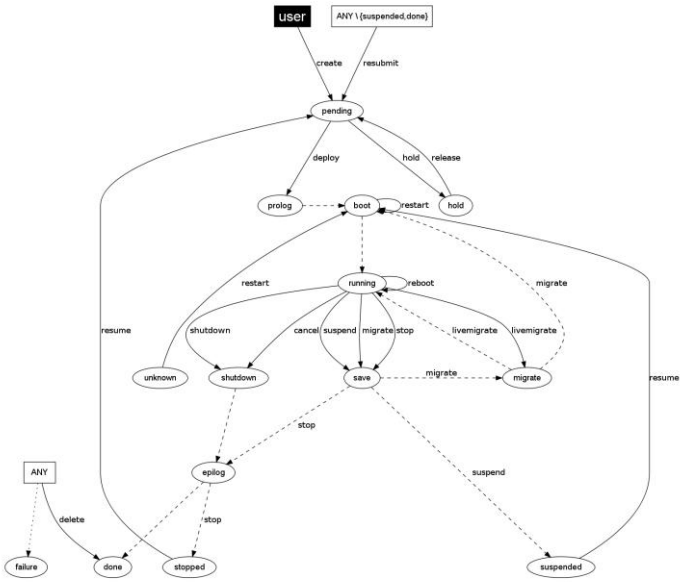

Figura 3 Ciclo di vita delle macchine virtuali

5.3 Eucalyptus

Eucalyptus[17][21] (Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems), è una piattaforma software per l'implementazione di soluzioni clouds privati, pubblici e ibridi in modalità Infrastructure as a Service che offre la possibilità di utilizzare diverse interfacce compatibili con Amazon EC2 e i servizi S3. Esiste anche in versione commerciale a pagamento ma in questa tesi è stata analizzata quella open source. Come OpenNebula, è una applicazione fortemente scalabile e si integra abbastanza facilmente in una infrastruttura già esistente, non richiedendo particolare hardware specializzato. Le caratteristiche più rilevanti riguardano la modularità che ne consente una forte personalizzazione e l'alto grado di flessibilità di distribuzione dei suoi componenti all'interno dell'infrastruttura hardware. Altro punto di forza è il supporto ai principali hypervisors come Kvm, Xen e VMware e la compatibilità con le principali distribuzioni linux come Red Hat Enterprise, CentOS, Debian, Ubuntu e openSUSE. Lo sviluppo di questo software è sponsorizzato da “Eucalyptus Systems”.

Le funzionalità principali sono:

- Hosts and Virtualization: l’architettura “Hypervisor-agnostic” supporta svariate tecniche di virtualizzazione basate su hypervisor quali Xen, KVM e VMware, ottenendo una astrazione al Cloud in cui gli utenti possono facilmente descrivere, eseguire, terminare e riavviare le loro istanze di macchina virtuale;
- Storage and Images: il componente Walrus è uno storage manager, compatibile con il servizio Amazon S3, che permette la memorizzazione delle immagini, dei kernels e ramdisks, e dei dati degli utenti. Inoltre il sistema EBS (Elastic Block Store) permette ad amministratori ed utenti di aggiungere ed eliminare dispositivi a blocchi nelle proprie istanze come dei volumi tramite le tecnologie AoE (ATA-

over-Ethernet) e iSCSI (Internet SCSI);

- Networking: è possibile impostare differenti modalità di configurazione in modo da soddisfare molteplici esigenze e architetture di rete. In relazione alla modalità di rete scelta, il sistema "Elastic IPs" permette agli utenti di allocare, associare e rilasciare gli indirizzi pubblici assegnati in maniera dinamica alle macchine virtuali. Inoltre il "Security Group Management" permette la gestione di un firewall costituito da un insieme di regole applicate alle VMs e associate ad uno specifico gruppo di utenti;
- Interfaces & APIs: il controllo del sistema Cloud da parte di amministratori e utenti può essere effettuata sia attraverso un apposito insieme di strumenti a riga di comando chiamato "euca2ool" che tramite interfaccia grafica via web. Per quanto riguarda le API Eucalyptus è compatibile con AWS (Amazon Web Services);
- Users and Groups: la gestione degli utenti e dei gruppi può essere effettuata tramite interfaccia web. La funzione "SSH Key Management" permette l'impiego di chiavi pubbliche e private per validare l'identità degli utenti quando effettuano il login sulle VMs tramite ssh. Gli utenti possono aggiungere e cancellare la coppia di chiavi da utilizzare.

I componenti principali sono, vedi fig. 4:

- Cloud Controller (CLC): recupera e gestisce le informazioni relative alle risorse disponibili nel Cloud tramite il Cluster Controller. E' inoltre, il punto di ingresso per amministratori e utenti di cui gestisce le credenziali di autenticazione;
- Cluster Controller (CC): schedula le macchine virtuali in esecuzione e ne monitora le informazioni principali;
- Node Controller (NC): è la macchina fisica in cui vengono eseguite le VMs;
- Walrus: è il componente di storage nel quale vengono memorizzate le immagini delle VMs;
- Storage Controller (SC): è un componente di storage opzionale e gestisce i dischi virtuali che eventualmente possono essere aggiunti alle istanze delle VMs.

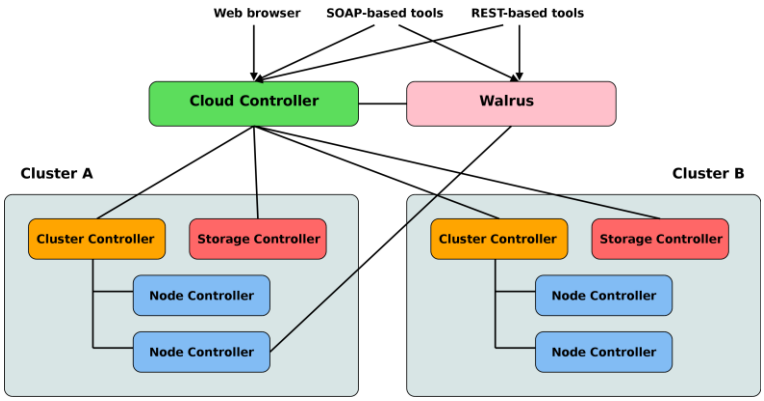

Figura 4 Schema delle componenti di Eucalyptus

5.4 Utilizzo e miglioramento di OpenNebula e Eucalyptus

L'approfondimento del funzionamento del Cloud in generale ha consentito di

individuare le soluzioni software che meglio si adattano alle caratteristiche ed esigenze dell'infrastruttura esistente. La sperimentazione effettuata su OpenNebula e Eucalyptus ha consentito di estendere le funzionalità del cluster migliorando ulteriormente il processo di razionalizzazione delle risorse già intrapreso con la virtualizzazione, come descritto nel capitolo 4, e inoltre ha permesso al consorzio CyberSAR di ampliare il suo campo di ricerca e disporre di un offerta di servizi ancora più vasta ed in linea con le attuali tecnologie di calcolo distribuito. Le funzionalità offerte dalle due applicazioni, sono state utilizzate sia come front-end per i servizi di Cloud che dalla Digital Library del progetto E-Pistemetec di cui CyberSAR fa parte. I test effettuati sul funzionamento complessivo del sistema di Cloud configurato, hanno evidenziato alcuni difetti e problematiche che rendevano il suo utilizzo poco vantaggioso e immediato, ad esempio è stato riscontrato che:

- I tempi di attesa per l'esecuzione delle macchine virtuali e la possibilità di accesso da parte dell'utente sono risultati spesso troppo lunghi;
- Il tipo di hypervisor adoperato per la virtualizzazione può vincolare e limitare la possibilità di personalizzare la configurazione delle immagine da inserire nel sistema cloud da parte degli utenti;
- La reattività e la responsività delle VMs, in relazione al carico della macchina fisica e all'hypervisor utilizzato sono risultate talvolta non accettabili;
- In relazione ai sistemi operativi "guest" e "host" utilizzati e alla posizione dei dati sull'hard disk si sono riscontrate prestazioni non soddisfacenti nelle operazioni di I/O.

Di seguito vengono descritti alcuni dei miglioramenti apportati durante la fase di test alle due applicazioni utilizzate. In questo modo è stata migliorata l'esecuzione di alcune operazioni e sono state aggiunte alcune funzionalità non immediatamente disponibili. Per alcuni particolari e maggiori dettagli concernenti i risultati ottenuti, si rimanda alla lettura dell'articolo "Enhancing Eucalyptus Community Cloud" riportato nell'appendice A, di cui il lavoro di questa tesi fa parte.

5.4.1 Ottimizzazione delle risorse

Per ottimizzare l'utilizzo delle risorse hardware, si è scelto di ricorrere, per quanto possibile, all'utilizzo di macchine virtuali. Escludendo i nodi di esecuzione delle VM, che per ovvie ragioni necessitano delle risorse di una macchina fisica, il resto delle funzionalità è stato virtualizzato. Infatti è stato testato che le prestazioni di una VM sono in grado di sostenere il carico di lavoro, relativamente basso, a cui le componenti delle due applicazioni Cloud utilizzate sono sottoposte. In particolare su OpenNebula il front-end, contenente l'installazione di tutte le componenti, è installato su una VM con sistema operativo Linux (Debian Lenny). Mentre per quanto riguarda Eucalyptus, le componenti: cloud, cluster, walrus e storage sono state installate su VMs con sistema operativo Linux (RedHat CentOS).

5.4.2 Kvm come Hypervisor

La scelta di usare QEMU/Kvm come hypervisor, porta notevoli vantaggi sia in termini di prestazioni ed efficienza delle macchine virtuali che in termini di portabilità. E'

possibile, infatti, utilizzare un normale Kernel standard sia lato host che lato guest, e usufruire dei miglioramenti derivati dallo sviluppo e ottimizzazioni del Kernel stesso. Il fatto che ormai i recenti servers supportino la virtualizzazione direttamente in hardware rende ancora più immediato l'utilizzo di questo hypervisor, semplicemente tramite l'operazione di caricamento, in base all'architettura utilizzata, dei moduli "kvm_intel" o "kvm_amd". I test eseguiti per misurare le prestazioni su operazioni di I/O, impostando l'uso della para-virtualizzazione con i driver "virtio" per le interfacce di rete e per i dischi, hanno riportato ottimi risultati. Sia OpenNebula che Eucalyptus supportano QEMU/Kvm interfacciandosi con la libreria "libvirt" ma è necessario modificare alcune impostazioni nei file di configurazione principali. In OpenNebula occorre editare il file "oned.conf":

```
#-------------------------------------------------------------------------------
# KVM Information Driver Manager Configuration
#   -r number of retries when monitoring a host
#   -t number of threads, i.e. number of hosts monitored at the same time
#-------------------------------------------------------------------------------
IM_MAD = [
      name       = "im_kvm",
      executable = "one_im_ssh",
      arguments  = "-r 0 -t 15 kvm" ]

#-------------------------------------------------------------------------------
#  KVM Virtualization Driver Manager Configuration
#    -r number of retries when monitoring a host
#    -t number of threads, i.e. number of hosts monitored at the same time
#    -l <actions[=command_name]> actions executed locally, command can be
#        overridden for each action.
#        Valid actions: deploy, shutdown, cancel, save, restore, migrate, poll
#        An example: "-l migrate,poll=poll_ganglia,save"
#-------------------------------------------------------------------------------
VM_MAD = [
        name       = "vmm_kvm",
        executable = "one_vmm_exec",
        arguments  = "-t 15 -r 0 kvm",
        default    = "vmm_exec/vmm_exec_kvm.conf",
        type       = "kvm" ]
```

Nello specifico, il driver consiste dei seguenti file che eventualmente è possibili modificare per variare il suo comportamento di default:

- "/usr/lib/one/mads/one_vmm_exec": descrive il driver generico;
- "/var/lib/one/remotes/vmm/kvm": contiene i comandi per eseguire le operazioni previste;
- "/etc/one/vmm_exec/vmm_exec_kvm.conf": contiene i valori di default usati nel template per le macchina virtuali;
- "/var/lib/one/remotes/vmm/kvm/kvmrc": contiene le istruzioni che verranno effettuate prima del caricamento del driver. In questo modo sarà possibile eseguire task specifici o passare determinate variabili al driver.

Per quanto riguarda Eucalyptus, il cui hypervisor di default è Xen, oltre che impostare Kvm nel file di configurazione "eucalyptus.conf" dei cluster nodes:

```
# The hypervisor that the Node Controller will interact with in order
# to manage virtual machines.  Currently, supported values are 'kvm'
# and 'xen'.
HYPERVISOR="kvm"
```

bisogna abilitare l'uso di "virtio" come driver per la gestione della rete e dei dischi:

```
# The following three KVM-specific options determine whether the
# hypervisor uses Virtio for specific types of I/O with the VM.
# (To use VirtIO, the VM must have the appropriate drivers installed.)

# If "1", use Virtio for EBS (elastic block store) volumes
USE_VIRTIO_DISK="1"
# If "1", use Virtio for the root file system disk
USE_VIRTIO_ROOT="1"
# If "1", use Virtio for the network card
USE_VIRTIO_NET="1"
```

5.4.3 Formato delle immagini

Ogni qualvolta viene istanziata una VM, prima che questa sia utilizzabile dall'utente, l'immagine del disco virtuale a cui è associata deve essere copiata dal repository delle immagini al cluster node in cui viene eseguita. Il tempo necessario per eseguire questa operazione dipende dalla dimensione dell'immagine da copiare. Per evitare questa attesa è stato usato il formato QCOW2 (QEMU Copy On Write), supportato da QEMU per la memorizzazione dei dischi immagine. Una delle sue caratteristiche più interessanti è quella di poter associare un file incrementale QCOW2 ad una immagine RAW. In questo modo il file QCOW2, sostanzialmente vuoto, cresce nel tempo in base alle operazioni su di esso effettuate, lasciando l'immagine RAW intatta. Nella pratica questo si traduce nel poter avere per un unica immagine disco tante istanze diverse indipendenti e inoltre una forte riduzione dei tempi di istanziazione della VM. In OpenNebula, per poter usare una istanza con formato QCOW2 avente backing file una immagine RAW, è stato necessario modificare il comportamento del Transfer Manager Driver durante l'operazione di copia dell'immagine master. Ciò è stato possibile anche grazie alla struttura altamente modulare del driver e alle diverse operazioni che questo riesce ad eseguire in relazione al modello di storage configurato in OpenNebula, vedi fig. 5.

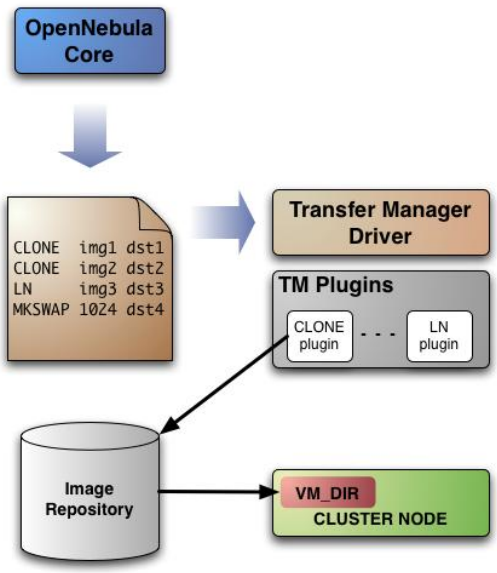

Figura 5 Schema di funzionamento del Transfer Manager Driver

Gli scripts presenti in "/usr/lib/one/tm_commands/", sono facilmente personalizzabili e definiscono le operazioni, per ogni tipo di storage supportato, che il driver può effettuare. Nella nostra configurazione i files disponibili sono:

```
tm_clone.sh
tm_ln.sh
tm_mkimage.sh
tm_mkswap.sh
tm_delete.sh
tm_mv.sh
```

Ogni script riceve come argomento almeno l'origine o la destinazione o entrambe nella seguente forma:

```
<host>:<path>
```

le azioni possibili sono le seguenti:

- CLONE: esegue una copia dell'immagine da “origine” a “destinazione”;
- LN: crea un link simbolico su “destinazione” che punta a “origine”;
- MKSWAP: genera una immagine swap su “destinazione” con dimensione in Mbyte pari a quella di “origine”;
- MKIMAGE: crea un immagine disco su “destinazione” e vi inserisce il contenuto specificato in “origine”;
- DELETE: cancella i file e directory specificate in “origine”;
- MV: sposta “origine” in “destinazione”;

Inoltre, affinché tali scripts funzionino, devono essere conformi ad alcune regole generali:

- restituire un valore di uscita pari a “0” o qualsiasi altro valore per indicare un errore;
- tutto ciò che viene scritto sullo “standard output” deve essere inserito sul corrispondente file di “log” della relativa macchina virtuale;
- in caso di errore, il messaggio deve essere scritto nello “standard error” nella seguente forma:
  ```
  ERROR MESSAGE --8<------
  error message here
  ERROR MESSAGE ------>8--
  ```

In “/usr/lib/one/mads/” è presente il file “tm_common.sh” che può essere utilizzato negli script descritti in precedenza nel seguente modo:

```
. /usr/lib/one/mads/tm_common.sh
```

In esso vi sono definite alcune funzioni utili per l'esecuzione di operazioni comuni, come:

- log: prende in ingresso come parametro il messaggio che verrà inserito nel file di “log” della macchina virtuale: `log "Creating directory $DST_DIR"`
- error_message: notifica un messaggio di errore, appositamente formattato, quando una azione non riesce ad essere eseguita: `error_message "File '$FILE' not found"`
- arg_host: restituisce la parte “hostname” da un parametro: `SRC_HOST=`arg_host $SRC``
- arg_path: restituisce la parte “percorso” da un parametro: `SRC_PATH=`arg_path $SRC``

- exec_and_log: esegue un comando e scrive nel "log". In caso di errore, verrà notificato nel file "oned.log" e lo script sarà terminato: **`exec_and_log "chmod g+w $DST_PATH"`**
- timeout_exec_and_log: simile al precedente ma il primo parametro considerato è il numero massimo di secondi destinati all'esecuzione dello script: **`timeout_exec_and_log 15 "cp $SRC_PATH $DST_PATH"`**

Le modifiche da apportare per l'utilizzo delle immagini in formato QCOW2, riguardano il file "tm_clone.sh", e sono riportate di seguito in neretto:

```
#!/bin/bash
SRC=$1
DST=$2

if [ -z "${ONE_LOCATION}" ]; then
    TMCOMMON=/usr/lib/one/mads/tm_common.sh
else
    TMCOMMON=$ONE_LOCATION/lib/mads/tm_common.sh
fi

. $TMCOMMON

SRC_PATH=`arg_path $SRC`
DST_PATH=`arg_path $DST`

SRC_HOST=`arg_host $SRC`
DST_HOST=`arg_host $DST`

log_debug "$1 $2"
log_debug "DST: $DST_PATH"

DST_DIR=`dirname $DST_PATH`

log "Creating directory $DST_DIR"
exec_and_log "$SSH $DST_HOST mkdir -p $DST_DIR" \
    "Error creating directory $DST_DIR"

case $SRC in
http://*)
    log "Downloading $SRC"
    exec_and_log "$SSH $DST_HOST $WGET -O $DST_PATH $SRC" \
        "Error downloading $SRC"
    ;;

*)
log "Creating qcow image of $SRC"
exec_and_log "$SSH $DST_HOST qemu-img create -f qcow2 -o backing_file=$SRC_PATH \
$DST_PATH" "Error copying $SRC to $DST"
    ;;
esac

exec_and_log "$SSH $DST_HOST chmod a+rw $DST_PATH"
```

In questo modo, piuttosto che eseguire la solita operazione di copia, in "/var/lib/one/<VM_ID>/images/" viene creato un file di tipo QCOW2 il cui backing file è l'immagine dell'istanza. Per verificare tali informazioni si può eseguire il comando:

```
# qemu-img info /var/lib/one/10/images/disk.0
image: /var/lib/one/10/images/disk.0
file format: qcow2
virtual size: 1.5G (1610612736 bytes)
disk size: 123M
cluster_size: 65536
backing file: /var/lib/one/images/535362ed8d9e2045e42457cfd9570f9e
```

Poiché in OpenNebula l'impostazione di default è il formato “RAW”, oltre alla modifica dello script per poter eseguire una istanza di una immagine in formato QCOW2, è necessario aggiungere nel template della macchina virtuale le seguenti righe nella sezione disk:

```
DISK   = [
         IMAGE_ID = 1,
         DRIVER = qcow2 # indica l'utilizzo di una immagine in formato QCOW2
       ]
```

In Eucalyptus, in maniera simile, è stato modificato i file “partition2disk” e “gen_kvm_libvirt_xml” presenti in “/usr/local/eucalyptus/usr/share/eucalyptus/”.

5.4.4 Prestazioni del Virtual disk

Sebbene il formato QCOW2 sia vantaggioso per gli aspetti descritti in precedenza, il suo utilizzo è penalizzante per quanto riguarda le prestazioni delle VMs nelle operazioni di scrittura sui dischi virtuali. L'esecuzione di alcuni test effettuati sulle VMs, hanno dimostrato che la velocità in scrittura di un file di 1 GB suddiviso in 1024 blocchi da 1 MB ciascuno su filesystem Linux ext3, è stata di 33 MB/s su un disco virtuale in formato RAW e 19 MB/s se il disco è in formato QCOW2. Questi valori sono da confrontare con le prestazioni della macchina host pari a 39 MB/s quando la scrittura è effettuata direttamente sul filesystem locale e 33 MB/s su di una immagine di disco virtuale in formato RAW. Eseguendo gli stessi test usando il formato QCOW2 con la pre-allocazione dei meta dati le prestazioni sono migliori, raggiungendo i 32 MB/s. Il problema sta nel fatto che la versione attuale di QEMU/KVM non supporta nel formato QCOW2 la possibilità di combinare l'utilizzo del backing file con la pre-allocazione dei meta-dati, tale funzionalità è in fase di implementazione sulle prossime versioni. Uno dei fattori che limita le prestazioni sul disco virtuale è l'operazione di allocazione del nuovo spazio disco, infatti, eseguendo una ri-scrittura del file da 1 GB scritto in precedenza, la velocità aumenta in maniera consistente raggiungendo approssimativamente i 40 MB/s per entrambi i formati immagine RAW e QCOW2. Un altro aspetto importante da considerare per meglio valutare i risultati dei test, è la posizione in cui verranno scritti i dati all'interno del disco, infatti si verifica un decremento della velocità di scrittura man mano che ci si sposta dai cilindri esterni del disco verso quelli interni. Le prestazioni del formato QCOW2 potrebbero essere un problema dato che anche le prestazioni di I/O di un disco di una macchina virtuale non sono eccezionali. Bisogna anche considerare che una grossa attività di scrittura nel filesystem di root delle macchine virtuali non è in ogni caso raccomandato e neppure necessario dal momento che è possibile aggiungere on-the-fly ulteriori volumi alle VMs, rilevati come dei normali dischi. I volumi, infatti, sono esportati dallo storage controller e importati nell’host che esegue la VM tramite protocollo iSCSI via rete, in modo da essere aggiunti alle VMs emulando un dispositivo PCI. Se le prestazioni dello storage controller sono buone e la larghezza di banda della rete è ragionevole, anche le prestazioni del volume saranno accettabili: l’esecuzione di un test di un file da 1 GB effettuato su un volume ha raggiunto 24 MB/s nel guest, da confrontare con i 25MB/s nell’host. Connettendo lo storage controller ad un NAS o SAN ad alte prestazioni, potrebbero essere raggiunte velocità più elevate. Nella tabella 1 vengono mostrate le prestazioni del virtual disk ottenute nei tests eseguiti:

Test description	PM (MB/s)	VM (MB/s)
ext3 on phys. dev. (sda3 local)	39	-
ext3 on phys. dev. (sda4 local)	30	29
ext3 on “raw” image (sda3 local)	33	33
ext3 on lvm “raw” image (iSCSI)	25	24
ext3 on lvm “raw” image (overwrite)	16	16
ext3 on “raw” image (overwrite)	-	41
ext3 on “qcow2” image (sda3 local)	-	19
ext3 on “qcow2” image (overwrite)	-	42
ext3 on “qcow2” preall. image (sda3 local)	-	32
ext3 on “qcow2” preall. image (overwrite)	-	42

Tabella 1 Prestazioni del virtual disk ottenute nei tests eseguiti

5.5 Installazione di OpenNebula

L'installazione può essere effettuata sia dai sorgenti dopo averli scaricati e compilati che tramite i pacchetti previsti per diverse distribuzioni linux (RHEL/CentOS, Debian, openSUSE, Ubuntu). I requisiti per l'installazione del front-end sono i seguenti:

- ruby >= 1.8.7 (Ruby Libraries Requirements)
- gcc (GNU project C and C++ compiler)

Bisogna inoltre installare le seguenti librerie di sviluppo ruby, utilizzate da OpenNebula: sqlite3, mysql client, curl, libxml, libxslt, ruby, expat. Per installare i vari “gems” si può usare lo script fornito in OpenNebula oppure procedere manualmente.

```
# wget http://rubyforge.org/frs/download.php/75573/rubygems-1.8.12.tgz
# tar zxvf rubygems-1.8.12.tgz
# cd rubygems-1.8.12
# ruby setup.rb all
# /usr/share/one/install_gems
```

Una volta soddisfatti i requisiti è possibile installare OpenNebula, per comodità, direttamente dai pacchetti, poiché in linea con l'ultima versione attualmente disponibile:

```
# apt-get install opennebula
```

Dopo l'installazione le diverse componenti di OpenNebula rispetto al filesystem vengono collocate secondo il seguente schema, vedi fig. 6:

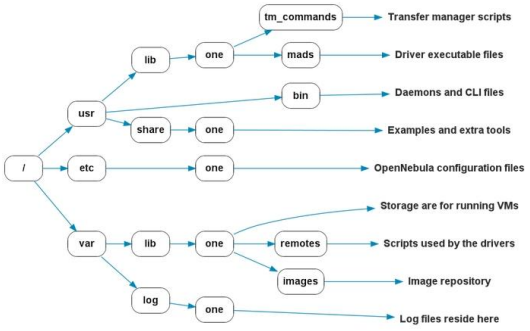

Figura 6 Installazione estesa di OpenNebula

Riguardo i cluster nodes, sono stati installati su macchine fisiche con sistema operativo linux (CentOS 6). E' stata scelta una CentOS piuttosto che una Debian per ragioni di compatibilità con il filesystem distribuito GPFS, prevedendo in futuro di usarlo al posto di NFS. A differenza del front-end, i nodi non necessitano di alcuna componente OpenNebula ma solamente dell'installazione e configurazione di uno tra gli hypervisors supportati (KVM, Xen, Vmware) e dei seguenti requisiti:

- ssh server in esecuzione
- ruby >= 1.8.7
- hypervisor

Durante la fase di configurazione e di gestione/amministrazione di OpenNebula non è necessario l'uso dell'utente root ma è consigliabile l'utilizzo di un utente speciale creato da OpenNebula in fase di installazione. Nel front-end quindi esiste un nuovo utente chiamato "oneadmin" e appartenente ad un nuovo gruppo chiamato "oneadmin". Il front-end deve poter comunicare con se stesso ed i cluster nodes tramite ssh senza che venga richiesta ogni volta la password di login. E' necessario creare su questi ultimi, a livello di sistema operativo, un nuovo utente "oneadmin" facendo in modo che user-id (uid) e group-id (gid) siano identici a quelli del front-end:

```
# groupadd --gid gid oneadmin
# useradd --uid uid -g oneadmin -d /path oneadmin
```

Per evitare la richiesta della password, si devono generare le chiavi ssh pubblica e privata per l'utente "oneadmin" nel front-end:

```
$ ssh-keygen
```

e poi inserire la chiave pubblica nel file "authorized_keys":

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

La stessa cosa andrà fatta dei cluster nodes. Alcune distribuzioni richiedono inoltre il settaggio di specifici permessi affinche l'autenticazione possa funzionare:

```
$ chmod 700 ~/.ssh/
$ chmod 600 ~/.ssh/id_dsa.pub
$ chmod 600 ~/.ssh/id_dsa
$ chmod 600 ~/.ssh/authorized_keys
```

E' importante sapere che OpenNebula gestisce i suoi utenti in maniera indipendente dal sistema operativo. Tali utenti sono definiti da una "username" e una "password", identificati da un id univoco e devono appartenere ad uno specifico gruppo gestito anch'esso da OpenNebula. Non è necessario creare un nuovo account unix per ogni nuovo utente OpenNebula perchè questi sono concetti slegati tra loro. Il sistema di autenticazione si basa sull'uso di una stringa di sessione controllata dal core di OpenNebula ogni volta che un utente OpenNebula esegue una qualsiasi operazione. Esistono due tipi di utenti OpenNebula:

- **oneadmin user:** questo account ha i privilegi sufficienti per eseguire tutte le operazioni possibili sul Cloud e viene creato la prima volta che OpenNebula viene mandato in esecuzione in base al contenuto del file "/var/lib/one/auth". Questo file contiene una sola linea in cui è indicato lo username e la password dell'utente "oneadmin" nella forma: "username:password". Se il file non esiste, viene cercato in "$HOME/.one/one_auth". Se non viene trovato neanche lì, OpenNebula non potrà lavorare correttamente poiché richiesto dal core, dalla CLI e dagli altri

componenti Cloud.

- **Regular user:** questo tipo di accounts devono essere creati dall'utente “oneadmin” ed hanno i privilegi necessari per le gestione degli oggetti (Templates, Images, Virtual Networks, etc) di cui sono proprietari. Possono però usare oggetti di altri utenti purché siano pubblici e appartengano allo stesso gruppo.

Per modificare o ripristinare le credenziali di autenticazione dell'utente “oneadmin”, eseguire i seguenti comandi:

```
$ mkdir ~/.one
$ echo "oneadmin:password" > ~/.one/one_auth
$ chmod 600 ~/.one/one_auth
```

Un gruppo in OpenNebula rende possibile l'isolamento degli utenti e delle risorse. In generale un utente può vedere e usare solo le risorse pubbliche appartenenti al suo gruppo. Come impostazione di default vengono creati i gruppi “oneadmin” che permette a tutti gli utenti appartenenti ad esso di eseguire qualsiasi operazione al pari dell'utente “oneadmin” ed il gruppo “user” che è utilizzato per tutti gli utenti senza particolari privilegi. Per la gestione degli utenti OpenNebula e dei relativi gruppi ci sono appositi comandi, ad esempio:

```
$ oneuser create regularuser password # crea un nuovo utente
$ oneuser passwd 1 password # modifica la password all'utente con id 1
$ oneuser list # visualizza la lista degli utenti
$ oneuser show regularuser # visualizza i dettagli di un utente
$ onegroup create "new group" # crea un nuovo gruppo
$ oneuser chgrp -v regularuser "new group" # cambia il gruppo di appartenenza di
un utente
$ onegroup list # visualizza la lista dei gruppi
```

A questo punto si può verificare l'installazione e far partire il servizio OpenNebula, come utente “oneadmin”, eseguire il comando:

```
$ one start
```

Se il servizio si è avviato correttamente si dovrebbero visualizzare in esecuzione il processo principale “oned” ed i suoi sotto-processi:

```
$ ps -edaf | grep one

/usr/bin/oned -f
/usr/bin/mm_sched -p 2633 -t 30 -m 300 -d 30 -h 1
ruby /usr/lib/one/mads/one_vmm_exec.rb -t 15 -r 0 kvm
ruby /usr/lib/one/mads/one_im_exec.rb -r 0 -t 15 kvm
ruby /usr/lib/one/mads/one_tm.rb tm_shared/tm_shared.conf
ruby /usr/lib/one/mads/one_hm.rb
ruby /usr/lib/one/mads/one_image.rb fs -t 15
```

5.5.1 Configurazione

OpenNebula viene fornito con una configurazione base di default fortemente personalizzabile. Questo è un aspetto che, se da un lato rende questo sistema di Cloud computing robusto per certi aspetti e adatto ad un uso in ambienti molto eterogenei, dall'altro richiede buone conoscenze dal punto di vista sistemistico e di networking, rendendo questo toolkit meno immediato rispetto ad altri software della sua categoria. Per usufruire delle funzionalità più interessanti, infatti, è necessario installare “from scratch” e configurare manualmente alcuni suoi sotto-sistemi fondamentali. L'architettura modulare di OpenNebula comprende l'uso di diversi drivers inclusi nell'installazione, in modo da

implementare scenari differenti che permettono ad esempio l'utilizzo di un filesystem condiviso o non condiviso piuttosto che uno strato software LVM. Ancora più interessante è la possibilità di estendere o creare nuovi drivers per realizzare soluzioni differenti ancora non previste o non incluse nel software. I file per il controllo e la gestione di una macchina virtuale sono eseguiti dal driver "Transfer Manager" e memorizzati in "/var/lib/one/<vm-id>", dove la subdirectory <vm-id> contiene l'immagine del disco dell'istanza. Il file di configurazione principale è "oned.conf" che insieme ai templates permettono di definire, tramite un insieme di attributi, una determinata caratteristica o funzionalità. Come per qualsiasi installazione e configurazione è buona norma visualizzare costantemente i diversi file di "log" di OpenNebula scritti in "/var/log/one/" durante tutte le sue fasi di esecuzione. A tal proposito si può impostare in "oned.conf" la verbosità desiderata secondo la seguente convenzione:

```
DEBUG_LEVEL: 0 = ERROR, 1 = WARNING, 2 = INFO, 3 = DEBUG
```

Hypervisor: come descritto in precedenza OpenNebula non viene fornito con alcun tipo di sistema di virtualizzazione ma spetta all'amministratore installarne uno tra quelli attualmente supportati nei cluster nodes. Si è scelto di usare KVM perché, a parte possedere un hardware con il supporto alla virtualizzazione, non necessità di moduli specifici aggiuntivi. Inoltre, fornisce una tecnica di virtualizzazione completa in cui ogni macchina virtuale è capace di interagire direttamente con il proprio hardware virtualizzato. Anche in questo caso è stata utilizzata la versione pacchettizzata disponibile su CentOS, i cui requisiti sono:

- CPU with VT extensions
- libvirt >= 0.4.0
- kvm kernel modules (kvm.ko, kvm-{intel,amd}.ko)
- the qemu user-land tools

```
# yum install qemu-kvm.x86_64 qemu-kvm-tools.x86_64 qemu-img.x86_64
libvirt.x86_64 libvirt-devel.x86_64
```

E' necessario installare anche la libreria "libvirt" poiché è utilizzata da OpenNebula per interagire con KVM in modo da gestire le macchine virtuali eseguendone lo start, lo stop e eventuali migrazioni tra hosts. E' stato necessario, inoltre, modificare il file "/etc/libvirt/qemu.conf" in modo da assegnare correttamente utenti e permessi ai vari file e processi utilizzati:

```
# The user ID for QEMU processes run by the system instance
user = "oneadmin"
# The group ID for QEMU processes run by the system instance
group = "oneadmin"
# Whether libvirt should dynamically change file ownership
# to match the configured user/group above. Defaults to 1.
# Set to 0 to disable file ownership changes.
dynamic_ownership = 0
```

e il file "/etc/libvirt/libvirtd.conf":

```
# Set the UNIX domain socket group ownership. This can be used to
# allow a 'trusted' set of users access to management capabilities
# without becoming root.
#
# This is restricted to 'root' by default.
unix_sock_group = "kvm"
#
# Set the UNIX socket permissions for the R/O socket. This is used
```

```
# for monitoring VM status only
#
# Default allows any user. If setting group ownership may want to
# restrict this to:
unix_sock_ro_perms = "0777"
#
# Set the UNIX socket permissions for the R/W socket. This is used
# for full management of VMs
#
# Default allows only root. If PolicyKit is enabled on the socket,
# the default will change to allow everyone (eg, 0777)
#
# If not using PolicyKit and setting group ownership for access
# control then you may want to relax this to:
unix_sock_rw_perms = "0777"
# Set an authentication scheme for UNIX read-only sockets
# By default socket permissions allow anyone to connect
#
# To restrict monitoring of domains you may wish to enable
# an authentication mechanism here
auth_unix_ro = "none"
#
# Set an authentication scheme for UNIX read-write sockets
# By default socket permissions only allow root. If PolicyKit
# support was compiled into libvirt, the default will be to
# use 'polkit' auth.
#
# If the unix_sock_rw_perms are changed you may wish to enable
# an authentication mechanism here
auth_unix_rw = "none"
```

I cluster nodes devono avere il demone di “libvirt” in esecuzione, ma per impedire che questa libreria al suo avvio possa impostare automaticamente alcune regole “iptables”, è stato eliminato il link presente all'interno di “/etc/libvirt/qemu/networks/autostart/”. L'utente “oneadmin”, che accede ai cluster nodes durante le operazioni di OpenNebula, deve appartenere ai gruppi “libvirt” e “kvm” in modo da poter usare tale demone per mandare in esecuzione le macchine virtuali. Oltre a queste impostazioni si consiglia la disabilitazione di SELinux, attiva di default nelle distribuzioni CentOS, modificando il file “/etc/selinux/config” con la seguente opzione: “SELINUX = disabile”

Host Monitoring: dopo aver configurato correttamente l'hypervisor il passo successivo consiste nell'impostare l'IM (Information Manager) di OpenNebula, in modo che nella comunicazione tra front-end e cluster nodes vengano usati i driver di virtualizzazione corretti, in questo caso quelli relativi a KVM (configurazione di default). A questo punto bisogna aggiungere gli hosts che verranno utilizzati come cluster nodes. Come detto in precedenza, un cluster node è un host registrato sul front-end di OpenNebula con la capacità di eseguire una macchina virtuale. I requisiti per la loro corretta gestione sono i seguenti:

- avere un account “oneadmin”;
- l'utente “oneadmin” deve poter autenticarsi tramite ssh in tutti i cluster nodes e nel front-end senza richiesta di password;
- risolvere, tramite DNS o il file “/etc/hosts” i nomi di tutti i cluster nodes e del front-end;
- avere uno degli hypervisor supportati da OpenNebula installato e correttamente configurato.

Avendo utilizzato KVM come hypervisor in una configurazione che condivide soltanto

il repository delle immagini, è necessario registrare gli hosts usando i seguenti parametri: “im_kvm” come Information Driver Manager, “vmm_kvm” come Virtualizzation Driver Manager e “tm_ssh” come Transfer Driver Manager:

```
$ onehost create host01 im_kvm vmm_kvm tm_ssh
```

L'esecuzione di questo comando provvederà, inoltre, a copiare in ”/var/tmp/one” del cluster node appena registrato gli scripts persenti in “/var/lib/one/remotes” presenti nel front-end, utilizzati per il monitoring sullo stato e le risorse disponibili.

Storage: il driver che si occupa della gestione dello storage è chiamato TM (Transfer Manager) e la configurazione di default prevede l'uso di un filesystem condiviso tra front-end e cluster nodes. Può essere usato un filesystem distribuito come NFS, GPFS, LUSTRE o simili, ma è essenziale che tutti gli hosts, front-end compreso, montino nello stesso percorso le directory in cui saranno memorizzate, vedi fig. 7:

- le immagini (master images) usate dalle macchine virtuali che verranno istanziate, creando di fatto un Image Repository. Il percorso di default è: “/var/lib/one/images”;
- la directory delle istanze che funge da cache per le macchine virtuali, identificata dalla variabile VM_DIR in “oned.conf”, percorso dove il TM driver colloca il file immagine clonato per le macchine virtuali in esecuzione.

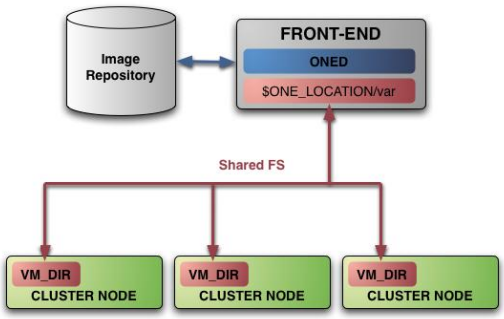

Figura 7 Repository delle immagini con condivisione delle sole istanze

Uno storage condiviso riduce il tempo di deployment delle macchine virtuali e rende possibile l'operazione di live migration, che consiste nello spostamento di macchine virtuali in esecuzione senza tempi di attesa. Per contro, in situazioni di uso intensivo del disco le prestazioni delle istanze potrebbero degradare. In questa sperimentazione si è scelto di condividere solamente il repository delle immagini tramite il filesystem NFS, mentre le istanze delle immagini vengono memorizzate localmente nei cluster nodes, vedi fig. 8.

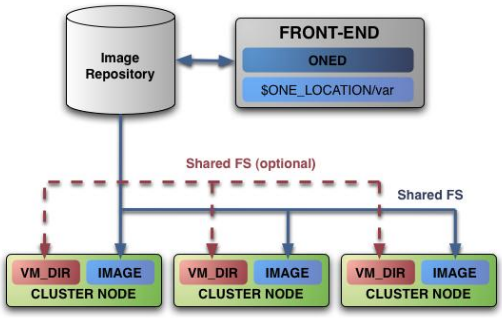

Figura 8 Configurazione e usata in questa sperimentazione

Per far funzionare questa configurazione, si deve impostare il TM driver in modo che l'operazione di clonazione dell'immagine di partenza avvenga tramite ssh, modificando il file “oned.conf”: in questo modo:

```
#-------------------------------------------------------------------------------
# SSH Transfer Manager Driver Configuration
#-------------------------------------------------------------------------------
TM_MAD = [
    name       = "tm_ssh",
    executable = "one_tm",
    arguments  = "tm_ssh/tm_ssh.conf" ]
```

Inoltre, dobbiamo modificare lo script “tm_ln.sh”, il quale normalmente crea un link in <VM_DIR> che punta all'immagine originale quando non è richiesta l'operazione di clone in modo che venga creata una directory remota ed un link nel cluster node di destinazione. Le modifiche da apportare sono riportate di seguito in neretto:

```
#!/bin/bash

SRC=$1
DST=$2

. /usr/lib/one/mads/tm_common.sh

SRC_PATH=`arg_path $SRC`
DST_PATH=`arg_path $DST`
DST_HOST=`arg_host $DST`

DST_DIR=`dirname $DST_PATH`

log "Creating directory $DST_DIR"
exec_and_log "ssh $DST_HOST mkdir -p $DST_DIR"

log "Link $SRC_PATH"
exec_and_log "ssh $DST_HOST ln -s $SRC_PATH $DST_PATH"
```

In maniera simile si può modificare il file “tm_clone.sh”, in modo che le diverse operazioni vengano eseguite direttamente nel cluster node di destinazione. E' sufficiente aggiungere ad ogni operazione il comando “ssh” sull'host remoto:

```
#!/bin/bash

SRC=$1
DST=$2

. /usr/lib/one/mads/tm_common.sh

SRC_PATH=`arg_path $SRC`
DST_PATH=`arg_path $DST`
DST_HOST=`arg_host $DST`

log "$1 $2"
log "DST: $DST_PATH"

DST_DIR=`dirname $DST_PATH`

log "Creating directory $DST_DIR"
exec_and_log "ssh $DST_HOST mkdir -p $DST_DIR"
exec_and_log "ssh $DST_HOST chmod a+w $DST_DIR"

case $SRC in
http://*)
    log "Downloading $SRC"
    exec_and_log "ssh $DST_HOST wget -O $DST_PATH $SRC"
    ;;
```

```
*)
    log "Cloning $SRC_PATH"
    exec_and_log "ssh $DST_HOST cp $SRC_PATH $DST_PATH"
    ;;
esac
exec_and_log "ssh $DST_HOST chmod a+w $DST_PATH"
```

Networking: Il sotto-sistema di rete in OpenNebula deve poter permettere ai processi in esecuzione nel front-end di accedere ai cluster nodes in modo da gestire e monitorare gli hypervisors e spostare i file delle immagini. Per quanto riguarda questo aspetto, nel front-end è stata configurata una interfaccia fisica sulla rete interna privata dedicata al Cloud. Sul front-end è necessaria anche una seconda interfaccia con indirizzamento pubblico, in modo da permettere l'accesso via web alla GUI Sunstone per le operazioni di gestione e configurazione da parte di amministratori e utenti. Per fare in modo che le macchine virtuali comunichino fra loro tramite una rete privata interna e siano raggiungibili da reti esterne a quella del sistema Cloud, è necessario fornire loro connettività di rete tramite i cluster nodes in cui sono eseguite. Nella configurazione di default ciò è reso possibile tramite un collegamento ethernet bridging, basato sulla creazione di bridges nei cluster

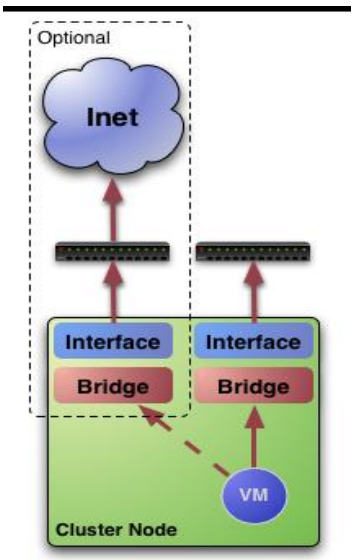

Figura 9 Collegamento ethernet bridging tra cluster nodes e macchina virtuale

node, vedi fig. 9.

La configurazione usata in questa sperimentazione prevede l'uso di due reti fisiche, una dedicata agli IP pubblici e l'altra alle virtual LANs private. In maniera identica in ogni clutser nodes, sono stati creati e configurati due bridges, nei quali vengono aggiunte le interfacce fisiche relative alla rete privata o interna agli hosts e a quella pubblica con connettività su Internet:

```
# brctl show
bridge name bridge id         STP enabled interfaces
br1         8000.001e682f02ac  no          eth1 # rete privata
br2         8000.001e682f02ad  no          eth2 # rete pubblica
```

Dopo aver impostato in maniera opportuna la configurazione di rete lato front-end e lato cluster nodes, in OpenNebula è necessario editare e registrare, tramite il template a

disposizione, tutte le tipologie di rete che si intende usare. Nel template è possibile settare una vasta gamma di parametri che consentono di adattare la configurazione della VNet alle proprie esigenze. La configurazione di rete privata utilizzata è la seguente (private.net):

```
NAME = "Private LAN"
TYPE = FIXED

# This vnet can be only used by the owner user
PUBLIC = NO

#Now we'll use the cluster private network (physical)
BRIDGE = br1

LEASES = [IP=10.10.1.1, MAC=20:01:00:00:00:01]
LEASES = [IP=10.10.1.2, MAC=20:01:00:00:00:02]
LEASES = [IP=10.10.1.3, MAC=20:01:00:00:00:03]
LEASES = [IP=10.10.1.4, MAC=20:01:00:00:00:04]
LEASES = [IP=10.10.1.5, MAC=20:01:00:00:00:05]

#Custom Attributes to be used in Context
GATEWAY = 10.10.1.254
DNS     = 10.10.1.254
```

mentre per la rete pubblica (public.net):

```
NAME    = "Pubblic LAN"
TYPE    = FIXED
PUBLIC  = Yes

# We have to bind this network to 'br2' for Internet Access
BRIDGE  = br2

LEASES  = [IP=130.10.0.1, MAC=50:20:20:20:20:21]
LEASES  = [IP=130.10.0.2]
LEASES  = [IP=130.10.0.3]
LEASES  = [IP=130.10.0.4]

# Custom Attributes to be used in Context
GATEWAY = 130.10.0.1
DNS     = 130.10.0.1
```

Dopo aver creato i diversi template per le reti da utilizzare bisogna registrarli su OpenNebula:

```
$ onevnet create private.net # registra il template private.net
ID: 0

$ onevnet create public.net # registra il template public.net
ID: 1
```

E' possibile visualizzare tutte le reti registrate con i relativi dettagli di configurazione come ad esempio il proprietario della rete (USER), il numero di IP-MACs assegnati alle macchine virtuali (LEASES) ed il bridge a cui è collegata (BRIDGE):

```
$ onevnet list # visualizza le reti disponibili
ID     USER         GROUP  NAME         TYPE   BRIDGE       PUB    LEASES
0      oneadmin     users  Private LAN  F      br1          Yes    0
1      oneadmin     users  Public_LAN   F      br2          Yes    0
```

Gli utenti OpenNebula possono visualizzare tutte le VNets appartenenti al loro gruppo. E' possibile condividere le VNets rendendole utilizzabili nelle istanze da altri utenti dello stesso gruppo, gestendo il flag “public” tramite il comando:

```
$ onevnet publish <vnetid> # pubblica una vnet
$ onevnet unpublish <vnetid> # pubblica una vnet
```

Nella nostra configurazione di rete finale, è stato usato e configurato un DHCP sulla macchina fisica che ospita il front-end. Le modifiche da apportare riguardano solamente i template di definizione delle reti pubblica e privata poiché devono essere coerenti con il mapping tra ciascun indirizzo IP e corrispondente MAC così come usato dal DHCP.

5.5.2 Istanziazione di VMs

Dopo aver configurato i diversi sottosistemi in base alle nostre esigenze, prima di poter eseguire il deploy di una istanza di macchina virtuale sul sistema di cloud è necessario creare i templates per la registrazione delle immagini che si intende utilizzare e per la definizione dell'istanza stessa. Ci sono tre differenti tipi di immagine:

- **OS**: contiene una immagine del sistema operativo correttamente funzionante, in ogni template di istanza è necessario definire un disco che si riferisce a questo tipo di immagine;
- **CDROM**: contiene una immagine di dati in sola lettura, in ogni template di istanza è possibile usarne solo una;
- **DATABLOCK**: contiene una immagine usata come storage per i dati, che possono essere acceduti e modificati da diverse macchine virtuali. Questo tipo di immagini possono essere create da dati già esistenti oppure come dispositivi vuoti.

Per registrare una immagine in OpenNebula si deve editare un file in cui sono presenti le informazioni riguardanti il nome con cui verrà registrata, il percorso da dove verrà importata ed il tipo. Ad esempio con il seguente file “image.one” definiamo il template per l'immagine di una debian squeeze:

```
# image import template
NAME = debian_squeeze
PATH = "/nfs/one/one-templates/debian_squeeze.img"
TYPE = OS
DESCRIPTION = "linux debian squeeze"
```

che può essere registrata eseguendo il comando:

```
$ oneimage create image.one
ID: 0
```

Invece con il seguente file “disk.one”, si crea un disco vuoto, poiché non viene specificato il percorso sorgente, di tipo “datablock” con dimensione 10GByte:

```
NAME          = "disk_datablock"
TYPE          = DATABLOCK
# No PATH set, this image will start as a new empty disk
SIZE          = 10240
FSTYPE        = ext3
PUBLIC        = NO
DESCRIPTION   = "Storage for my Thesis experiments."
```

In entrambi i casi il comando “oneimage create <file>”, non fa altro che copiare il file immagine dalla sorgente indicata nel template al repository delle immagini assegnandoli un id. Nel caso di un disco di tipo “datablock” come quello dell'esempio precedente, verranno usati i comandi “dd” e “mkfs” rispettivamente per la creazione e formattazione del disco secondo le specifiche indicate nel template. Per visualizzare le immagini registrate e tutti i relativi dettagli si può usare il comando “oneimage” con i soliti parametri “list” o “show”.

Nel templates per la descrizione dell'istanza sono previste diverse sezioni in cui è

possibile utilizzare numerosi parametri. Di seguito viene riportato il template "debian_squeeze.one" che crea una istanza dell'immagine importata in precedenza avente id = 1, per la quale si richiede un hardware virtuale con CPU = 1, RAM = 512 (Mbyte), un disco aggiuntivo di tipo "datablock" con id = 2 e una scheda di rete con id = 0:

```
# Capacity Section
NAME   = debian_squeeze
CPU    = 1
MEMORY = 512

# Disks Section
DISK   = [ IMAGE_ID = 1 ]

# DATABLOCK image, mapped to hdb
DISK   = [ IMAGE_ID  = 2,
           TARGET    = hdb
         ]

# Network Section
NIC    = [ NETWORK_ID = 0 ]
```

Il deploy di questo template può essere mandato in esecuzione con il comando:

```
$ onevm create debian_squeeze.one
```

per visualizzare le macchine virtuali correntemente istanziate da tutti gli utenti si può usare:

```
$ onevm list a
```

mentre per ottenere le informazioni relative ad una istanza:

```
$ onevm show <VM_ID>
```

Non appena viene fatto il deploy l'immagine viene copiata dal repository delle immagini al cluster node in cui viene eseguita, creando il file disk.0 in "/var/lib/one/<VM_ID>/images/". Questa operazione, effettuata dallo script "tm_clone.sh" del Transfer Manager, può essere più o meno lunga in relazione alla dimensione del file da clonare, in questa fase lo stato della macchina virtuale sarà "PENDING". Viene quindi eseguito il boot dell'immagine copiata e se non si verificano errori lo stato della macchina virtuale passerà da "BOOT" a "RUNNING".

Il template utilizzato precedentemente è molto semplice ma per far eseguire macchine virtuali con caratteristiche diverse, sono disponibili numerosi parametri da usare nelle apposite sezioni.

Sezione "CAPACITY", definisce le risorse di calcolo:

- **name:** imposta il nome dall'istanza, se non viene fornito ne verrà generato uno di default con il seguente formato: one-<VID>;
- **memory:** indica la quantità di memoria RAM in Mbyte da destinare all'istanza;
- **cpu:** indica la percentuale di cpu da destinare all'istanza. Il valore 1 assegna tutta la cpu mentre con valori inferiori è possibile assegnarne solo una frazione di essa;
- **vcpu:** indica il numero di cpu virtuali da destinare all'istanza, il valore di default è 1;

Sezione "OS", definisce le caratteristiche del sistema:

- **arch:** indica l'architettura della cpu da virtualizzare, valori tipici sono "i686" default per KVM o "x86_64".

- **boot:** indica il dispositivo da usare per effettuare il boot dall'istanza, valori tipici sono “hd” usato di default, “fd”, “cdrom” o “network”, corrispondenti rispettivamente alle opzioni “-boot [a | c | d | n]” del comando KVM.

Sezione “DISK”, definisce gli hard disk di una istanza. E' possibile definire tante sezioni per quanti dischi si ha necessità di inserire. Un disco può essere aggiunto ad una istanza in due modi: usando una immagine dal repository delle immagini oppure usando un qualsiasi file disco senza che debba essere registrato nel repository. Uno dei limiti di KVM è quello di poter supportare solo 4 dispositivi IDE, quindi nel caso si voglia istanziare una macchina virtuale con un numero maggiore di dischi, è necessario ad esempio l'utilizzo di dispositivi SCSI. Inoltre, poiché OpenNebula in fase di contestualizzazione di una ISO userà il dispositivo “hdc” e, se non specificato diversamente il dispositivo “hde” per le immagini di tipo “datablock”, si incorrerà in un errore anche con meno di 4 dischi. E' raccomandato l'uso del parametro “TARGET” per indicare il mapping del dispositivo, ad esempio “TARGET = hdb”:

- **image_id:** indica l'id dell'immagine da usare come disco;
- **bus:** indica il tipo di dispositivo disco da emulare, corrisponde all'opzione “if” dell'argomento “-driver” del comando KVM. Valori tipici sono: “ide”, “scsi” o “virtio”;
- **driver:** indica il formato del disco immagine, corrisponde all'opzione “format” dell'argomento “-driver” del comando KVM. Valori tipici sono “raw” di default e “qcow2”;
- **target:** indica il dispositivo nel quale mappare il disco;
- **type:** indica il tipo di dispositivo da assegnare all'istanza, corrisponde all'argomento “-driver” del comando KVM. Il valore di default è “disk”, altri valori sono: “fs”, “swap”, “cdrom” o “floppy”. Il tipo “fs” e “swap” possono essere aggiunti nell'istanza on-the-fly;
- **source:** indica il percorso sorgente del file disco o una “url”, può essere usato solo per dischi di tipo “floppy”, “disk”, “cdrom” e “block”;

Sezione “NIC”, definisce le interfacce di rete. E' possibile definire tante sezioni per quante interfacce si ha necessità di inserire:

- **target:** indica il nome del dispositivo “tun: creato per la macchina virtuale, corrisponde all'opzione “ifname” dell'argomento “-net” del comando KVM;
- **model:** indica il modello di hardware ethernet da emulare. Valori tipici sono: “rtl8139”, “e1000”, “pcnet” e “virtio”. Una lista dei modelli disponibili si può ottenere con il seguente comando:

  ```
  # kvm -net nic,model=? -nographic /dev/null
  ```

- **script:** indica il nome dello script shell da eseguire dopo la creazione del dispositivo “tun”, corrisponde all'opzione “script” dell'argomento “-net” del comando KVM.

Sezione “I/O DEVICES”, definisce ulteriori funzionalità:

- **PAE**: (Physical Address Extension) è una modalità che permette alle macchine virtuali a 32 bit, di indirizzare più di 4GByte di memoria;

- **ACPI**: (Advanced Configuration and Power Interface) parametro indispensabile per la gestione dello spegnimento e riavvio delle macchine virtuali;
- **VNC: (**Virtual Network Computing) fornisce la possibilità di controllare in maniera remota una macchina virtuale.

```
# pae and acpi features
FEATURES  =  [
   pae    = { yes | no },
   acpi   = { yes | no }
   ]

# vnc configuration
GRAPHICS = [
   type    = "vnc",
   listen  = "0.0.0.0",
   port    = "5901",
   keymap  = "en-us"
   ]
```

In OpenNebula le immagini create di default hanno la caratteristica di essere di tipo non persistente con la conseguenza che tutte le operazioni eseguite dall'istanza verranno perse al suo spegnimento o shutdown. Infatti, come descritto precedentemente, durante l'esecuzione di una macchina virtuale, l'immagine master da cui essa viene istanziata non viene modificata ma bensì clonata dal repository al cluster node che la esegue in <VM_DIR>/<VM_ID>/images/ creando così il file "disk.0". Per ovviare a questo comportamento, dopo un regolare shutdown della macchina virtuale:

```
$ onevm shutdown <vm_id>
```

è possibile salvarne il contenuto usando un apposito comando che farà una copia del file "disk.0" da <VM_DIR>/<VM_ID>/images/ al repository delle immagini creandone una nuova:

```
$ onevm saveas <vmid> <disk_id> <image_name>
```

Esiste però un altra interessante possibilità, e cioè quella di creare immagini persistenti in modo da poter ripartire dall'ultima modifica eseguita anche dopo lo spegnimento dell'istanza, ad esempio con il seguente comando:

```
$ oneimage persistent <image_id>
```

In una configurazione che condivide sia le immagini che le istanze, piuttosto che fare la copia dell'immagine come descritto in precedenza, nel cluster node che esegue l'istanza viene creato un link simbolico in <VM_DIR>/<VM_ID>/images/ che punta al repository delle immagini. L'ulteriore vantaggio è quello di avere un deployment immediato dell'istanza senza che sia necessaria alcuna ulteriore operazione per salvare nuovamente il disco quando la macchina virtuale viene spenta. Lo svantaggio principale è causato dal fatto che l'istanza scrive direttamente dentro l'immagine master di partenza, se per qualche motivo i dati vengono corrotti o persi non sarà più possibile usare tale immagine. Inoltre, su tale tipo di immagine, non è possibile poter istanziare più di una macchina virtuale per volta, per questo le immagini persistenti non possono essere rese pubbliche. Una macchina virtuale può essere terminata oltre che con un regolare shutdown con la sua cancellazione forzata e conseguente distruzione:

```
$ onevm delete <vm_id>
```

in questo caso verrà cancellato anche il file "disk.0" e non essendosi verificato nessun caso in cui viene effettuata l'operazione di copia dell'immagine come descritto in precedenza,

non sarà più possibile recuperarne il contenuto. In maniera simile anche le operazioni di sospensione o stop di una macchina virtuale in esecuzione non prevedono l'operazione di copia.

5.5.3 Contextualization

Usando una immagine ISO, è possibile passare dei parametri ad una nuova macchina virtuale. Affinché questa ISO venga correttamente montata all'avvio di una nuova istanza, quest'ultima dovrà essere appositamente personalizzata in modo che in fase di boot vengano eseguiti determinati script. La procedura è indipendente dal sotto-sistema di rete perciò può essere usata anche per la configurazione delle interfacce di rete. Una sua applicazione può essere quella di associare due dischi alla macchina virtuale in cui nel primo è presente il sistema operativo dell'istanza mentre nel secondo c'è una ISO, montata nell'istanza come partizione, contenente i file usati in fase di contestualizzazione. Ad esempio l'immagine ISO potrebbe contenere i seguenti file, vedi fig. 10:

- context.sh: viene incluso per default e contiene uno script bash, in cui OpenNebula potrà inserire alcune variabili in base ai parametri specificati nel template della macchina virtuale;
- init.sh: script bash, chiamato all'avvio dalla macchina virtuale per la configurazione dei servizi specifici dell'istanza;
- certificates: directory che contiene i certificati per qualche servizio;
- service.conf: configurazione di determinato servizio

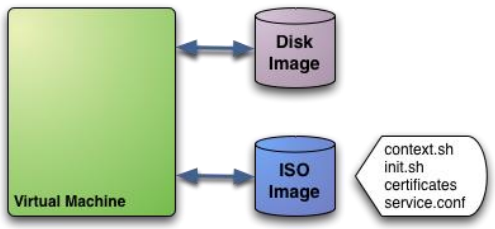

Figura 10 Esempio di contestualizzazione

Le modifiche da apportare al template della macchina virtuale riguardano la definizione di una sezione “CONTEXT” in cui è presente la specifica dei contenuti del file ISO (files e directories) che verranno scritti dentro la macchina virtuale per il loro uso successivo. Alcuni attributi che si possono utilizzare, anche se non obbligatori, hanno un significato particolare:

- FILES: indica i files e le directories che verranno incluse nell'immagine ISO;
- TARGET: indica, in relazione al sistema operativo presente nell'istanza, il dispositivo nel quale sarà resa disponibile l'immagine ISO. In assenza di questo attributo, ne verrà generato automaticamente uno di default di tipo “hdb” o “sdb” corrispondentemente al prefisso impostato nel file “oned.conf”.

Le variabili all'interno della sezione “CONTEXT”, tradotte da OpenNebula e aggiunte al file “context.sh” in fase di contestualizzazione, possono essere specificate in modalità hardcoded:

```
hostname = "NAME_HOST"
```

oppure, recuperando quelle usate nei diversi templates di OpenNebula (virtual machines, virtual networks, images e users):

```
ip_gen = "10.0.0.$VMID" # $<template_variable>
```

oppure usando le variabili predefinite:

```
$UID # id del proprietario della macchina virtuale

$TEMPLATE # l'intero template in formato XML codificato in base64
```

Di seguito viene mostrato un esempio, specifico per una immagine basata su Debian, che imposta l'hostname e l'indirizzo IP dell'istanza, crea un nuovo utente e inserisce la chiave pubblica ssh per consentirne l'accesso senza che venga richiesta la password.

```
CONTEXT = [
          files       = "/vm_context/init.sh /vm_context/id_rsa.pub",
          target      = "hdc",
          ip_public = "$NIC[IP, NETWORK=\"Public\"]",
          hostname    = "$NAME",
          username    = "user01"
     ]
```

Nel front-end, quindi, dovrà esserci una directory “vm_context” con all'interno: la chiave pubblica “id_rsa.pub” da aggiungere alle chiavi autorizzate per il nuovo utente e lo script “init.sh” che eseguirà il lavoro di configurazione. L'istanza viene configurata per montare la ISO all'avvio ed eseguire “init.sh” tramite lo script “/etc/rc.local”. In questo modo si potrà modificare il funzionamento dello script, più agevolmente in maniera locale piuttosto che all'interno della macchina virtuale. Lo script “rc.local” potrebbe essere qualcosa del tipo:

```
#!/bin/sh -e

mount -t iso9660 /dev/sdc /mnt

if [ -f /mnt/context.sh ]; then
  . /mnt/init.sh
fi

umount /mnt

exit 0
```

mentre lo script “init.sh” legge i parametri presenti in “context.sh” e prosegue con la configurazione:

```
#!/bin/bash

if [ -f /mnt/context.sh ]; then
  . /mnt/context.sh
fi

hostname $HOSTNAME
ifconfig eth0 $IP_PUBLIC

useradd -m $USERNAME

mkdir -p ~$USERNAME/.ssh
cat /mnt/id_rsa.pub >> ~$USERNAME/.ssh/authorized_keys

chown -R $USERNAME /home/$USERNAME
```

Per problemi di sicurezza, in ogni file relativo al Transfer Manager Driver è stata aggiunto un parametro che in base al valore della variabile “SECURE_CONTEXT” presente in “/etc/one/tm_shared/tm_sharedrc”, può rendere più o meno permissiva la

provenienza dei file inseribili nell'immagine ISO. Modificando il suo valore di default, si potranno inserire solamente il file "context.sh" ed eventuali urls di tipo http:

```
# Set to 1 to disable adding files to context image other than context.sh itself
or http url's
SECURE_CONTEXT=0
```

5.5.4 OpenNebula Sunstone

E' una interfaccia grafica accessibile via web, che permette sia agli amministratori che ai normali utenti l'esecuzione di tutte le operazioni per la gestione dell'infrastruttura Cloud. Il suo comportamento e funzionalità possono essere estese tramite appositi plugins. I browsers attualmente supportati sono Firefox (> 3.5) e Google Chrome. Anche se è possibile installare Sunstone su un server qualsiasi, è stato configurato nel front-end. Per il suo funzionamento, dopo aver installato e correttamente configurato OpenNebula, l'unico ulteriore requisito è la presenza di alcuni pacchetti ruby:

```
# gem install json rack sinatra thin
```

Nel file di configurazione "sunstone-server.conf" che si trova in "/etc/one/", seguendo la sintassi tipica di YAML, è necessario indicare alcuni parametri come:

- l'indirizzo IP pubblico del server sul quale il servizio Sunstone viene eseguito in modo da potervi accedere anche da remoto;
- la porta in cui il server rimane in ascolto, per default è la 9869;
- il metodo di autenticazione usato. Se non specificato diversamente, verrà usato il metodo "basic auth", basato sulle credenziali già registrate nel database di OpenNebula.

Per far partire il servizio, è sufficiente eseguire il comando:

```
$ sunstone-server start
```

il cui file di "log" è "/var/log/one/sunstone.log". In Sunstone vi è anche la possibilità di monitorare graficamente diverse informazioni riguardanti i cluster nodes e le macchine virtuali in uso, semplicemente avviando il servizio di accounting:

```
$ oneacctd start
```

Il livello di dettaglio dipende dalla configurazione del demone "oneacctd" e può essere modificata editando il file "acctd.conf" in "/etc/one/". Una volta terminata la fase di configurazione sarà possibile accedere alla schermata di login collegandosi, con uno dei browser supportati, all'indirizzo pubblico specificato in precedenza: http://onecc.ca.infn.it:9869/, vedi fig. 11.

Figura 11 Interfaccia grafica Sunstone

5.5.5 EC2

Il web service EC2 Query fornisce la possibilità di eseguire e gestire le proprie macchine virtuali su OpenNebula tramite l'omonima interfaccia fornita da Amazon usando un qualsiasi tool che la supporta. Il web service è implementato sopra lo strato OpenNebula Cloud API (OCA) che espone tutte le funzionalità del Cloud, usando "Thin" e "Sinatra" come framework web, vedi fig. 12.

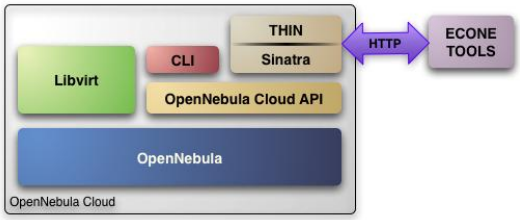

Figura 12 Econe Tools

L'implementazione attuale include le operazioni di registrazione e upload di una immagine e di esecuzione, terminazione e descrizione di una istanza. Per il suo funzionamento, dopo aver installato e correttamente configurato OpenNebula, bisogna installare alcuni pacchetti ruby:

```
# gem install amazon-ec2 # amazon-ec2 Query API librery
# gem install sinatra thin # web framework and thin web server
# gem install curb sqlite3-ruby # libraries for the Client Tools
# apt-get install libsqlite3-ruby libcurl4-gnutls-dev libopenssl-ruby1.8 #
libraries for the Client Tools
```

E' necessario modificare il file di configurazione di OpenNebula "oned.conf" in modo da abilitare il driver relativo a EC2:

```
#-------------------------------------------------------------------------------
#  EC2 Information Driver Manager Configuration
#-------------------------------------------------------------------------------
IM_MAD = [
      name       = "im_ec2",
      executable = "one_im_ec2",
      arguments  = "im_ec2/im_ec2.conf" ]
#-------------------------------------------------------------------------------
```

Il file di configurazione specifico è "econe.conf" e si trova in "/etc/one/". Seguendo la sintassi tipica di YAML, è necessario indicare alcuni parametri come:

- il servizio xml-rpc del demone "oned", solitamente http://localhost:2633/RPC2;
- il server sul quale il servizio ec2 viene eseguito che insieme alla porta costituisce l'url da utilizzare per la connessione;
- la porta per le connessioni in ingresso, per default è la 4567;
- il metodo di autenticazione usato per il server "econe". Se non specificato diversamente verrà usato il metodo "EC2" basato sulla seguente convenzione: il valore per l'"AWSAcesskeyId" è l'username dell'utente su OpenNebula mentre l'"AWSSecretAccessKey" è l'hash della relativa password;
- Il tipo di macchina virtuale abilitato per gli utenti EC2 ed il relativo template.

Prima di poter usare il servizio occorre effettuare ulteriori operazioni di configurazione. E' raccomandata la creazione di un nuovo gruppo per la gestione degli utenti EC2:

```
$ onegroup create ec2
```

```
ID: 100
```

in modo da aggiungere gli utenti destinati a questo servizio a questo gruppo:

```
$ oneuser chgrp ec2-user_id 100
```

bisogna aggiungere alcune regole ACL per questo gruppo:

```
$ oneacl create "@100 VM+NET+IMAGE+TEMPLATE/* CREATE+INFO_POOL_MINE"
```

e abilitare tali utenti al deploy delle loro macchine virtuali sui cluster nodes, ad esempio la seguente regola:

```
$ oneacl create "@100 HOST/#1 USE"
```

abilita gli utenti appartenenti al gruppo EC2 con id 100 al deploy nell'host con id 1.

Bisogna, inoltre, definire una apposita virtual network con il comando "onevnet" in cui siano presenti gli indirizzi IP da assegnare alle istanze EC2. Questa Vnet dovrà avere il flag "public" attivo ed essere aggiunta al gruppo "user". Occorrerà poi aggiornare il template "m1.small.erb" presente in "/etc/one/ec2query_templates/", in modo che venga usata la Vnet definita in precedenza. Questo template è usato come default per le istanze di tipo EC2. Nella stessa directory è possibile aggiungere altri templates con nuovi "tagli" per le macchine virtuali creando un template per ciascuno di essi e poi inserendo in "econe.conf" il nome del nuovo taglio e del relativo template in questo modo:

```
# VM types allowed and its template file (inside templates directory)
:instance_types:
  :m1.small:
    :template: m1.small.erb
  :m1.large:
    :template: m1.large.erb
```

A questo punto è possibile far partire il servizio con il comando:

```
$ econe-server start
```

il cui file di "log" è "/var/log/one/econe-server.log".

Un utente ec2 potrà quindi eseguire in maniera remota dal suo pc, i comandi "econe" che gli consentono di fare l'upload di una immagine da un file in formato "img":

```
$ econe-upload -H -U http://onecc.ca.infn.it:4567 -K opennebula-user -S hash-
opennebula-password /path/to/image
Success: ImageId ami-00000001
```

registrare l'immagine su OpenNebula per abilitarne la sua istanziazione:

```
$ econe-register ami-00000001
Success: ImageId ami-00000001
```

visualizzare le immagini registrate o monitorare quelle in esecuzione:

```
$ econe-describe-instances -H -U http://onecc.ca.infn.it:4567 -K  opennebula-user
-S hash-opennebula-password
```

eseguire l'istanza dell'immagine:

```
$ econe-run-instances -H -U http://onecc.ca.infn.it:4567 -K opennebula-user -S
hash-opennebula-password ami-00000015
```

e terminare la sua esecuzione:

```
$ econe-terminate-instances -U http://onecc.ca.infn.it:4567 -K  opennebula-user -
S hash-opennebula-password
```

5.6 Installazione di Eucalyptus

Poiché le fasi della procedura di installazione e configurazione sono concettualmente abbastanza simili a quelle precedentemente descritte per OpenNebula, tali argomenti non verranno trattati in maniera così dettagliata ma si rimanda alla documentazione presente sul sito "http://open.eucalyptus.com". Verranno quindi illustrati i passaggi generali evidenziando le procedure specifiche effettuate per Eucalyptus. Sono state installate tutte le componenti di Eucalyptus e a ciascuna di esse è stata dedicata una macchina virtuale. Nei nodes controller, che richiedono una macchina fisica, è stato installato l'hypervisor Kvm, vedi fig. 13.

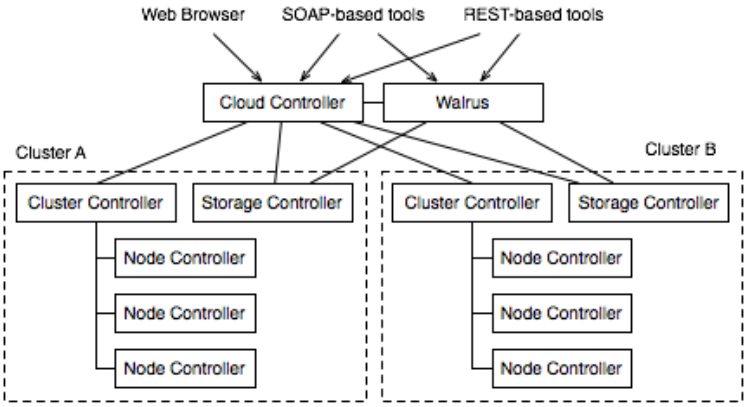

Figura 13 Componenti di Eucalyptus

Il software è disponibile in versione pacchettizzata per le distribuzioni supportate ma in questa sperimentazione si è scelto di installarlo su una "CentOS 6.0" tramite compilazione dei sorgenti, operazione anch'essa ampiamente documentata sul sito di riferimento. In questo caso quindi, i requisiti di partenza devono includere anche tutti gli strumenti necessari per la compilazione. Le prime modifiche da effettuare per l'esecuzione del software sono:

- disabilitazione di SELinux: è sufficiente modificare il file "/etc/selinux/config" impostando la seguente opzione: "SELINUX=disabile";
- creazione dell'utente e del gruppo "eucalyptus". Tale utente nei nodi in cui è presente la libreria "libvirt", dovrà essere aggiunto al gruppo "libvirt";
- nei nodes controller bisogna modificare il file di configurazione "/etc/libvirt/libvirtd.conf", in modo da impostare correttamente i permessi di lettura e scrittura utilizzati dalla libreria "libvirt" in maniera del tutto simile a quanto fatto per OpenNebula. Inoltre è necessario eliminare il link presente all'interno di "/etc/libvirt/qemu/networks/autostart/" per impedire l'impostazione automatica di alcune regole "iptables";
- impostare a 32 il numero di dispositivi di loopback presenti nel sistema.

A questo punto è possibile impostare alcune variabili d'ambiente con il comando:

```
# euca_conf --setup
```

eseguire lo script corretto in base al servizio Eucalyptus da avviare:

```
# /usr/local/eucalyptus/etc/init.d/eucalyptus-<service> start
```

abilitare/disabilitare i componenti negli hosts opportuni tramite il comando:

```
# /usr/local/eucalyptus/usr/sbin/euca_conf --enable <component>
```

Le varie componenti devono essere poi registrate nel front-end:

```
# /usr/local/eucalyptus/usr/sbin/euca_conf --register-walrus <front end IP
address>
# /usr/local/eucalyptus/usr/sbin/euca_conf --register-cluster <clustername>
<front end IP address>
# /usr/local/eucalyptus/usr/sbin/euca_conf --register-sc <clustername> <front end
IP address>
```

La presenza di eventuali problematiche di funzionamento possono essere verificate controllando i file di log in "/usr/local/eucalyptus/var/log/eucalyptus/".

5.6.1 Configurazione

Il file principale di configurazione è "eucalyptus.conf" in "/usr/local/eucalyptus/etc/eucalyptus/" nel quale vengono definite, per le diverse componenti, alcune delle informazioni per il suo corretto funzionamento. Di seguito vengono elencate le più importanti:

- percorso di installazione di Eucalyptus;
- abilitazione/disabilitazione del EBS (Elastic Block Storage);
- livello di verbosità dei log;
- la porta in cui sta in ascolto il CC;
- la lista dei nodi con cui il CC dovrà comunicare;
- l'hypervisor utilizzato nei NC;
- abilitazione/disabilitazione dei driver virtio;
- percorso della directory condivisa in cui vengono memorizzate le immagini delle VMs in esecuzione;
- tutte le informazioni relative alla configurazione generale della rete nelle diverse componenti, come: le interfacce di rete fisiche pubbliche e private utilizzate, i bridges per la comunicazione con le VMs, il percorso del server DHCP ecc.

I cluster nodes devono essere registrati nel front-end:

```
# /usr/local/eucalyptus/usr/sbin/euca_conf --register-nodes <Node IP address>
```

e poi possibile verificare i diversi tipi di VM e le risorse disponibili:

```
# euca-describe-availability-zones verbose
AVAILABILITYZONE        eclusteri       172.16.x.x
AVAILABILITYZONE        |- vm types     free / max   cpu   ram  disk
AVAILABILITYZONE        |- m1.small     0008 / 0021   1   1024    25
AVAILABILITYZONE        |- c1.medium    0003 / 0009   1   2048    25
AVAILABILITYZONE        |- m1.large     0003 / 0009   2   2048    25
AVAILABILITYZONE        |- m1.xlarge    0001 / 0003   2   4096    25
AVAILABILITYZONE        |- c1.xlarge    0001 / 0003   6   5850    25
```

5.6.2 Gestione delle immagini e istanziazione di VMs

Per poter aggiungere una immagine di macchina virtuale su Eucalyptus, l'amministratore o l'utente devono fornire, in file separati il kernel, il ramdisk e il

filesystem di root. Questi file dovranno poi essere registrati sulla componente walrus considerando il fatto che l'inserimento del kernel e ramdisk è consentito solo all'amministratore. Di seguito viene descritto un esempio usando i comandi forniti dagli "euca2ools", aggiunta del kernel:

```
# euca-bundle-image -i <kernel file> --kernel true
# euca-upload-bundle -b <kernel bucket> -m /tmp/<kernel file>.manifest.xml
# euca-register <kernel-bucket>/<kernel file>.manifest.xml
```

aggiunta del filesystem di root:

```
# euca-bundle-image -i <vm image file>
# euca-upload-bundle -b <image bucket> -m /tmp/<vm image file>.manifest.xml
# euca-register <image bucket>/<vm image file>.manifest.xml
```

aggiunta, opzionale, del ramdisk:

```
# euca-bundle-image -i <initrd file> --ramdisk true
# euca-upload-bundle -b <initrd bucket> -m /tmp/<initrd file>.manifest.xml
# euca-register <initrd bucket>/<initrd file>.manifest.xml
```

Le operazioni per eseguire il "bundle", associando il kernel e l'eventuale ramdisk con l'istanza, sono le seguenti:

```
# euca-bundle-image --kernel <eki-xxxxxxx> --ramdisk eri-<xxxxxxxx> -i <vm image
file>
```

Per richiedere una macchina virtuale, un utente, dopo essersi autenticato nel sistema tramite interfaccia web, deve interagire con il front-end utilizzando gli strumenti "euca2ools" per eseguire operazioni come la visualizzazione delle immagini disponibili:

```
# euca-describe-images
```

l'esecuzione di una istanza di VM, indicando l'immagine ed il tipo:

```
# euca-run-instances <emi-id> -t <vm-type>
```

visualizzazione delle istanze correntemente in esecuzione:

```
# euca-describe-instances
```

Dopo avere scelto un template tra quelli disponibili, la relativa immagine della VM verrà prelevata dal componente walrus, compattata nelle giuste dimensioni e poi trasferita in uno dei cluster nodes disponibili per essere eseguita dall'hypervisor. In questo host, l'hypervisor, comunicando con il front-end assegna alla VM un indirizzo MAC e IP virtuali assegnato ad esempio tramite DHCP, e ne esegue il boot. L'utente può a questo punto accedere alla VM tramite ssh. Per evitare che in uno stesso cluster node, ogni volta che viene richiesta una nuova istanza dello stesso tipo, vengano eseguite le operazioni descritte precedentemente, una copia del file immagine viene tenuto in una directory che assume la funzionalità di cache per le istanze. Ciò consente di accelerare notevolmente i tempi di avvio di una VM se l'immagine disco di questa è già presente in cache.

5.6.3 Aggiunta di un volume

Tramite il componente SC, Eucalyptus fornisce un servizio di storage compatibile con EBS di Amazon, che permette di aggiungere e rimuovere dischi nelle istanze. La modalità di aggiunta on-the-fly richiede nella VM in esecuzione che il modulo "acpiphp" sia correttamente caricato. Di seguito vengono forniti una serie di comandi da lanciare sul front-end. Per creare un volume bisogna indicarne la dimensione ed il cluster a di riferimento:

```
euca-create-volume -s 10 -z <cluster-name>
```

per visualizzare i volumi disponibili:

```
euca-describe-volumes
```

per eseguirne l'attach bisogna fornire l'id dell'istanza e specificare l'id del volume ed il target su cui mapparlo nella VM:

```
euca-attach-volume -i <i-xxxxxxxx> -d /dev/vdx <id-vol>
```

per eseguire il detach, specificare l'id del volume:

```
euca-detach-volume <id-vol>
```

Nella VM in esecuzione, se non si sono verificati problemi, eseguendo il comando "fdisk -l" sarà possibile visualizzare la presenza del nuovo volume.

6. Conclusioni

Esistono numerose modalità con cui le risorse di calcolo possono essere fornite agli utenti, tra cui il Grid e il Cloud computing trattati in questa tesi. Il Grid computing, dopo aver compiuto passi da gigante e riscosso in passato un discreto successo, oggi risente di qualche difficoltà dovuta alla poca flessibilità generale del sistema e alla sua complessità di implementazione. Il Cloud computing sta attualmente attraversando una fase di veloce sviluppo, di trasformazione e di forte interesse da parte delle comunità di sviluppatori e utenti. Il suo punto di forza principale è la grande capacità di adattamento all'infrastruttura di calcolo esistente, la possibilità di scalare rapidamente in base al carico di lavoro presente e di supportare le richieste di un grande numero di utenti che ottengono le risorse tramite la rete Internet. In questo lavoro si è dimostrata la facilità con cui si possa realizzare un sistema di calcolo distribuito, basato su Cloud computing usando dei framework open source, in una infrastruttura di calcolo già esistente come il polo CyberSAR di ingegneria. Openebula e Eucalyptus, si sono dimostrate ottime applicazioni per l'implementazione di sistemi di Cloud. L'evoluzione del calcolo distribuito in generale, consentirà di rispondere a domande come: quale sia l'architettura distribuita di virtualizzazione migliore per un sistema di Cloud computing; quali caratteristiche dovrà possedere uno scheduler di gestione di macchine virtuali per un uso sempre più efficiente delle risorse; come riuscire ad implementare configurazioni di rete sempre più flessibili, performanti e sicure; quali domini di applicazione potranno meglio avvantaggiarsi dai servizi offerti dal Cloud computing. Ulteriori possibili studi e sviluppi futuri di questo lavoro di sperimentazione, consisteranno nel testare le nuove funzionalità sviluppate nei due toolkit e la loro integrazione con altre tecnologie quali VirtualBox, Xen, e WMWare.

6.1. Comparazione tra Grid e Cloud computing

Il Grid computing può essere considerato come sistema di calcolo distribuito precursore del Cloud computing con il quale condivide un approccio orientato al servizio. Il passaggio tra le due tecnologie, che mirano entrambe ad aumentare la flessibilità e l'affidabilità delle risorse hardware cercando di ridurne i costi generali, deriva da un processo di evoluzione che sposta le finalità da una infrastruttura che offre storage e risorse di calcolo come è il caso della Grid ad una maggiormente focalizzata su aspetti economici che fornisce risorse e servizi astratti come è il caso del Cloud. Il termine Cloud computing non è quindi un nuovo nome per il Grid computing, ma si distingue da esso per diversi aspetti fondamentali[28]:

- Architettura: l'approccio Grid è basato su un middleware usato come mezzo per promuovere la condivisione e la federazione delle risorse comunque separate da domini amministrativi differenti, quello di un sistema Cloud è in genere gestito da una unica entità con autorità di amministrazione, configurazione e pianificazione;
- Modello di business: in un sistema Grid è “orientato al progetto” in cui gli utenti

acquistano un determinato numero di unità di servizio o ore di CPU. Nel Cloud tale modello è "orientato al consumo" in maniera molto simile a quanto avviene per la fornitura dell'energia elettrica, gas e acqua. Nel Cloud fornito da Amazon, ad esempio, l'utente paga per le risorse di cui realmente usufruisce come numero di CPU, quantità di RAM, larghezza di banda ecc;

- Gestione delle risorse: la richiesta di risorse da parte degli utenti in una Grid è gestita dal Local Resource Manager che si occuperà di schedulare i diversi job, parallelizzandoli quando possibile, in modo da avere poche istanze molto esose di risorse che vengono eseguite contemporaneamente. La richiesta può essere tale da saturare la quantità di risorse disponibili causando spesso lunghe code di esecuzione in cui i job degli utenti rimangono in attesa di essere eseguiti. Nel Cloud, esistono tecniche che minimizzano le attese per il rilascio delle risorse, cercando di migliorare la scalabilità del sistema e supportando il numero più alto di utenti. Il meccanismo di schedulazione che sta alla base dei sistemi Cloud deve essere più sofisticato, in quanto dovrà sincronizzare e condividere le medesime risorse hardware, per gestire un gran numero di piccole istanze eseguite contemporaneamente all'interno del sistema in real time;
- Modello delle applicazioni: generalmente la Grid supporta diversi tipi di applicazione che vanno dall'HPC (High Performance Computing) a jobs batch caratterizzati da tempi di esecuzione più o meno lunghi di tipo "compute intensive" o "data intensive". Sostanzialmente anche nei sistemi Cloud potrebbero essere eseguite applicazioni simili ad eccezione di quelle del tipo HPC che necessitano di una connessione di rete a bassa latenza per una efficiente comunicazione tra numerosi processori. Questa tecnologia di calcolo, è ancora relativamente troppo giovane per poter definire in maniera più specifica quali sono i tipi di applicazione per lei più idonei ma in generale dovranno essere orientate alle transazioni, poco dipendenti le une dalle altre e di tipo interattivo piuttosto che batch;
- Sicurezza: il problema relativo alla sicurezza è stato implementato nelle fondamenta dell'infrastruttura Grid con i protocolli GSI e CAS usati per l'autenticazione, autorizzazione e la protezione generale delle comunicazioni in ambito Grid. Questo approccio pur essendo dispendioso in termini di risorse permette di ottenere un ottimo sistema di accounting e auditing, garantendo un livello di sicurezza che previene i rischi di accessi non autorizzati. Attualmente il modello di sicurezza presente nei sistemi Cloud, essendo molto più semplice e meno sicuro, rappresenta uno dei punti deboli di più urgente miglioramento.

6.2. Comparazione tra OpenNebula e Eucalyptus

Pur essendo due applicazioni che permettono di realizzare la tipologia di cloud IaaS, nelle modalità privata, pubblica e ibrida, sono risultate sotto diversi aspetti abbastanza diverse[22][24][26][27]:

- Filosofia: Eucalyptus è il sistema di Cloud più conosciuto, usato anche da Ubuntu Enterprise Cloud e RedHat. La filosofia di Eucalyptus è quella di produrre una emulazione open-source di Amazon EC2 che utilizza il software "euca2ools" come implementazione dei relativi comandi ed un sistema di storage distribuito chiamato "walrus" molto simile a quello di Amazon S3. Ciò permette agli utenti che hanno familiarità con le tecnologie EC2 e S3 di poter usare Eucalyptus semplicemente

introducendo qualche comando e variabile d'ambiente. OpenNebula è orientato maggiormente verso un Cloud privato puro che offre all'utente la possibilità di utilizzare i relativi servizi disponibili;

- Installazione: Eucalyptus necessita di maggiori risorse hardware. Richiede, infatti, di almeno un host dedicato al controllo della componente cloud e cluster oltre che alla necessità di dover installare in ogni cluster node la parte di software specifico affinché questi possano comunicare con il Cluster/Cloud Controller. In OpenNebula invece è possibile concentrare tutte le componenti, cluster nodes a parte, in un unico server installando il software solamente nel front-end;
- Configurazione e amministrazione: uno dei punti di forza di OpenNebula è la sua architettura altamente modulare ed ampiamente personalizzabile che facilita l'integrazione con la maggior parte delle piattaforme di virtualizzazione e dei tool per il cloud esistenti. Questo è un aspetto che avvantaggia l'utente finale a cui è permesso, a differenza di Eucalyptus, poter creare templates personalizzati in cui si possono richiedere specifiche caratteristiche e risorse per la creazione di Vnets e VMs. Eucalyptus invece, mantiene una separazione molto più netta tra l'ambiente riservato agli amministratori e quello dei normali utenti, non è previsto che questi ultimi accedano direttamente al sistema, se non tramite l'utilizzo di interfaccia web. In generale in Eucalyptus si cerca di nascondere il più possibile la complessità del sistema riducendo le scelte disponibili all'utente a quelle già predefinite dall'amministratore mentre in OpenNebula l'utente dispone di una ampia scelta di parametri che per contro aumentano la possibilità di commettere errori di configurazione. Riguardo l'amministrazione tramite interfaccia grafica, esistono diverse possibilità per entrambe le applicazioni. Eucalyptus dispone di due plug-in per il browser Firefox: HybridFox e ElasticFox che facilitano e semplificano la gestione del cloud evitando l'uso della riga di comando. OpenNebula utilizza un sua interfaccia chiamata “Sunstone”, utilizzabile da amministratori e utenti, che oltre a permettere di eseguire tutte le operazioni di gestione e amministrazione, fornisce una serie di informazioni riguardanti l'accounting delle risorse, caratteristica non presente in Eucalyptus;
- Istanziazione immagini: nel complesso le operazione per l'esecuzione di una macchina virtuale risultano più semplici e meno dispendiose in termini di tempo per OpenNebula. Diversamente da Eucalyptus in cui si ha poca flessibilità sul tipo di VMs che si possono creare, l'operazione di trasferimento di una immagine nel sistema Cloud, oltre a non richiede l'inclusione dei file relativi al kernel e al ramdisk è molto meno macchinosa rispetto al lavoro di segmentazione, compressione e decompressione eseguito dal componente walrus;
- Condivisione dei dati: la configurazione di default di OpenNebula prevede la condivisione dei dati tramite il filesystem NFS che se da un lato risulta di facile configurazione dall'altro può rappresentare il collo di bottiglia quando si ha necessità di trasferire grosse mole di dati e diventare un problema riguardante la sicurezza poiché durante le operazioni di trasferimento i dati non vengono criptati. In Eucalyptus la possibilità di utilizzare il filesystem GPFS risolve in parte tali problematiche. Tale scelta, per contro, necessita di hardware aggiuntivo in cui installare il server GPFS, richiede un ulteriore lavoro di installazione e configurazione generale ed inoltre vincola la scelta del sistema operativo alle sole distribuzioni con esso compatibili.

BIBLIOGRAFIA

Citazioni di documenti tratti dal WWW (URL):

[1] Consorzio CyberSAR, http://www.cybersar.com

[2] GARR: la rete italiana dell'Università e della ricerca, http://www.garr.it

[3] Digital Library E-Pistemetec, http://www.e-pistemetec.net

[4] SystemImager, http://www.howtoforge.com/howto_linux_systemimager

[5] LDAP, http://www.openldap.org

[6] Ganglia, http://ganglia.info

[7] MRTG, http://www.enterastream.com/whitepapers/mrtg

[8] IGI, http://www.italiangrid.it

[9] ALICE, http://aliceinfo.cern.ch/Public/Welcome.html

[10] Globus, http://www.globus.org

[11] http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaat/liaatvirtualization.htm

[12] http://www.ibm.com/developerworks/library/l-hypervisor

[13] http://www.ibm.com/developerworks/library/l-linux-kvm

[14] http://www.ibm.com/developerworks/linux/library/l-qemu

[15] http://www.ibm.com/developerworks/library/os-cloud-anatomy

[16] http://opennebula.org

[17] http://open.eucalyptus.com

[18] C12G, http://www.c12g.com

[19] NIST, http://www.nist.gov/index.html

[20] Open Cloud Manifesto, http://www.opencloudmanifesto.org

Libri – Ebook – Articoli :

[21] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, Dmitrii Zagorodnov - The Eucalyptus Open-source Cloud-computing System

[22] Patrícia Takako Endo, Glauco Estácio Gonçalves, Judith Kelner, Djamel Sadok - A Survey on Open-source Cloud Computing Solutions

[23] Ian Foster, Carl Kesselman, Steven Tuecke - The Anatomy of the Grid: Enabling Scalable Virtual Organizations

[24] Peter Sempolinski, Douglas Thain - A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus

[25] Susanta Nanda, Tzi-cker Chiueh - A Survey on Virtualization Technologies

[26] Junjie Peng, Xuejun Zhang, Zhou Lei, Bofeng Zhang, Wu Zhang, Qing Li – Comparison of Several Cloud Computing Platforms

[27] Jonathan S Ward – A Performance Comparison of Clouds

[28] Ian Foster, Yong Zhao, Ioan Raicu, Shiyong Lu - Cloud Computing and Grid Computing 360-Degree Compared

[29] Ian Foster, Carl Kesselman, Steven Tuecke – Tha Anatomy of the Grid

Enhancing Eucalyptus Community Cloud

Andrea Bosin[1,2,3], Matteo Dessalvi[4], Gian Mario Mereu[3], Giovanni Serra[3]
[1]*Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Cagliari, I-09042 Monserrato, Italy*
[2]*Università degli Studi di Cagliari, Dipartimento di Fisica, I-09042 Monserrato, Italy*
[3]*Consorzio Cybersar, I-09123 Cagliari, Italy*
[4]*Università degli Studi di Cagliari, DRSI, I-09100 Cagliari, Italy*
E-mail: {andrea.bosin, matteo.dessalvi}@dsf.unica.it, {gmariomereu, giovanni.srr}@gmail.com
Received **** 2011

Abstract

In the last few years, the cloud computing model has moved from hype to reality, as witnessed by the increasing number of commercial providers offering their cloud computing solutions. At the same time, various open-source projects are developing cloud computing frameworks open to experimental instrumentation and study. In this work we analyze Eucalyptus Community Cloud, an open-source cloud-computing framework delivering the IaaS model and running under the Linux operating system. Our aim is to present some of the results of our analysis and to propose some enhancements that can make Eucalyptus Community Cloud even more attractive for building both private and community cloud infrastructures, but also with an eye toward public clouds. In addition, we present a to-do list that may hopefully help users in the task of configuring and running their own Linux (and Windows) guests with Eucalyptus.

Keywords: cloud, IaaS, Eucalyptus, KVM, qcow2

1. Introduction

According to NIST [1], "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". Such a definition is quite general, and is not bound to any specific enabling technology or hardware and software implementation.

Cloud resources are presented to users according to one of three different delivery or service models:

- *Software as a Service (SaaS).* The service provided to users is to employ the provider's applications running on a cloud infrastructure. The applications are accessible from different client devices through either a program interface or a thin client interface, such as a web browser. Users can at most manage limited user-specific application configuration.
- *Platform as a Service (PaaS).* The service provided to users is to deploy onto the cloud infrastructure their own applications developed using languages, libraries, and tools supported by the provider. Users have control over the deployed applications and possibly configuration settings.
- *Infrastructure as a Service (IaaS).* The service provided to users is to provision fundamental computing resources (processing, storage, networks, etc.) where users are able to deploy and run arbitrary software, such as operating systems and applications. Users have control over operating systems, storage, and deployed applications; and possibly limited control of network configuration (e.g., host firewalls or IP address reservation).

Depending on the way in which resources are organized and made available to users, we can distinguish between four different deployment models:

- *Private cloud.* The cloud infrastructure is provisioned for exclusive use by a single organization.
- *Community cloud.* The cloud infrastructure is provisioned for exclusive use by a specific com-

munity of users from organizations that have shared concerns.

- *Public cloud.* The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them.
- *Hybrid cloud.* The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability.

While private and community cloud infrastructures may be owned, managed, and operated by the organizations, a third party, or some combination of them, and may exist on or off premises, public clouds exist on the premises of the cloud provider.

The Open Cloud Manifesto [2], with its motto "dedicated to the belief that the cloud should be open", puts openness as one of the core principles of cloud computing, thus complementing the NIST model by envisioning a long-running perspective based on openness.

In this paper we analyze Eucalyptus Community [3], an open-source cloud-computing framework delivering the IaaS model and running under the Linux operating system. Eucalyptus Community exhibits openness in at least two important ways: in source code and in the adopted Amazon EC2 [4] and S3 [5] application programming interfaces (API), which have a public specification. The developers present Eucalyptus as "a framework that uses computational and storage infrastructure commonly available to academic research groups to provide a platform that is modular and open to experimental instrumentation and study" [3]. Openness in source code allows us to fully study, experiment, customize and enhance such a framework.

The aim of this work is to present the results of our analysis and to propose some enhancements that, in our opinion, can make Eucalyptus Community Cloud even more attractive for building both private and community cloud infrastructures, but also with an eye toward public clouds.

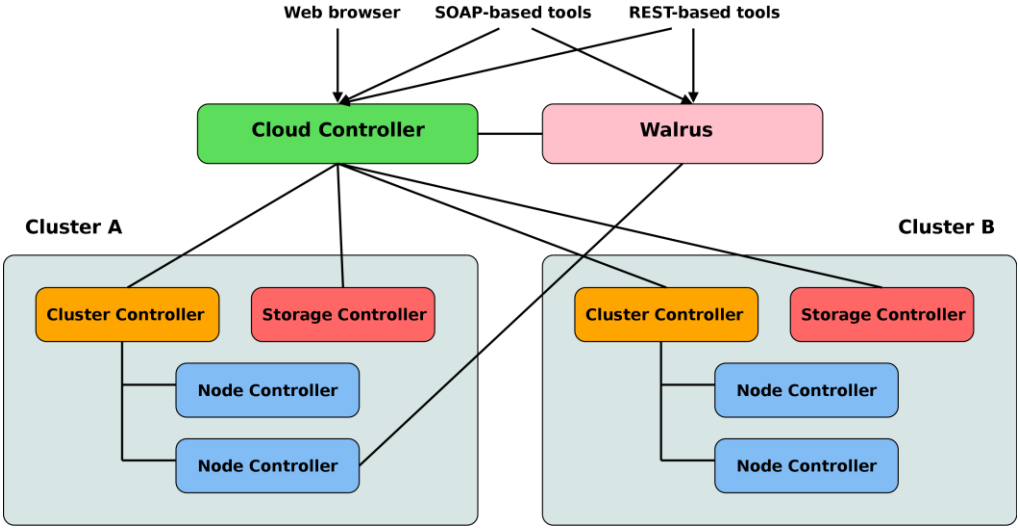

Figure 1. Eucalyptus Community Cloud architecture.

In Section 2 we present an overview of the Eucalyptus Community open-source cloud-computing framework, its design and its main functionality. Section 3 describes Eucalyptus out-of-the-box configuration, and points out some of its limitations. In Section 4 we discuss a number of enhancements aimed at mitigating some of the bottlenecks and some experiments showing how it is possible to take advantage of different underlying hardware and software resources. Section 5 tackles the problem of porting an external physical or virtual machine into Eucalyptus since, in our experience, this can be the nightmare of advanced users wishing to run their own virtual machine images. At last, Section 6 draws some conclusions.

2. Eucalyptus overview

In Eucalyptus Community, the IaaS delivery model is accomplished by providing virtual machines [6] to users: the framework provides a number of high level management services and integrates them with the lower level virtualization services found in many recent distributions of the Linux operating system. The system allows users to start, control, access, and terminate entire virtual machines. Virtual machines (VM or guests or instances) run somewhere on the physical machines (PM or hosts) that belong to the underlying physical cloud infrastructure.

Eucalyptus Community has a flexible and modular architecture with a hierarchical design as depicted in **Figure 1**. Each high level component is implemented as a stand-alone Web service [7]:

- **Node Controller** (NC) runs on a PM and controls the execution, inspection, and termination of VM instances on the host where it runs.
- **Cluster Controller** (CC) schedules and monitors VM execution on specific node controllers, as well as manages virtual instance network.
- **Storage Controller** (SC) is optionally associated to a cluster controller and is responsible for the management (allocation, use and deletion) of virtual disks that can be attached to VM instances; virtual disks (or volumes) are off-instance storage that persists independently of the life of an instance.
- **Walrus** is a put/get storage service, providing a simple mechanism for storing and accessing virtual machine images and user data.
- **Cloud Controller** (CLC) is the entry-point for users and administrators; it queries cluster controllers for information about resources (instances, images, volumes, etc.), makes high level scheduling decisions, and manages user credentials and authentication.

PMs running the node controller service are grouped into one or more independent clusters, where each cluster is managed by a cluster controller and optionally coupled to a storage controller; all cluster controllers are coordinated by one cloud controller, while one Walrus service provides storage for all virtual machine images and optional user data. User interaction involves the cloud controller for VM and volume management through the EC2 API, and Walrus for the management of VM images and user files through the S3 API; the access to VMs from the network (e.g. login via secure shell) is transparently granted or denied by the cluster controllers based on the security policies (or access groups) specified by users.

Users can interact with Eucalyptus Community either through command line tools such as euca2ools [8] or s3cmd [9] or web browsers plug-ins such as Hybridfox [10]. In addition, the CLC publishes a web interface for user registration and credentials generation.

3. Eucalyptus out-of-the-box configuration

Eucalyptus Community Cloud (ECC) provides a standard out-of-the-box configuration for the lower level virtualization services to allow a simple set-up of the cloud infrastructure. Such configuration affects mainly the node controllers which are responsible of the bare execution of VMs. The out-of-the-box configuration discussed here is by no means the only possible, as we are going to show in the next section, but it represents a simple and reasonable starting point. In this section we cover the details which we consider most relevant, and point out some of the correlated limitations and bottlenecks.

VMs run atop the *Xen* [11] hypervisor, even though ECC interacts with the specific hypervisor through an abstraction layer, i.e. the *libvirt* virtualization API [12]. Xen requires a special Xen-aware kernel both on the host and on the guest if para-virtualization [13] is used; only if the host hardware supports hardware-assisted virtualization (HVM), as it is the case with recent CPU [14-16], guest operating systems can be executed unmodified. As a consequence, if HVM is not enabled, Windows operating systems cannot be executed, while Linux operating systems must be booted from a Xen-aware kernel.

To run a VM instance, ECC needs separate image files for kernel, ramdisk and root filesystem, which are uploaded by users and retrieved by the system through the Walrus service; kernels and ramdisks can be registered with Walrus only by cloud administrators. This means that it is neither

possible to boot a VM from the (native) kernel and ramdisk in the root filesystem, nor to update external kernels and ramdisks without the intervention of cloud administrators. Windows can only be booted using a custom-made kernel/ramdisk combination. Walrus current implementation is extremely inefficient in managing large image files (say larger that 1 GB), since it performs many operations reading the files from and writing them to disk every time, instead of piping all the operations using the main memory and writing the result to disk only at the end.

The image file of the root filesystem partition is acquired immediately before starting a VM instance; a copy of the former (and also of the kernel and the ramdisk) is cached in a directory on the node controller in such a way that a new VM instance of the same type and on the same node controller will use the cached images, thus avoiding the overhead of retrieving them once again. The set-up of the root filesystem image is very time consuming; if it is not already present in cache, the image must be first retrieved from Walrus and saved on the node controller filesystem, then copied into the caching directory, and at last the VM is started. If the image is found in the node controller caching directory, a copy is performed and the VM is started. It is not unusual to have root filesystem large 5-10 GB in size, and the need for multiple copies slows down VM start-up times.

Figure 2 summarizes the main steps for the deployment of a VM: (1) a user requests a new VM to CLC, (2) CLC authorizes the request and forwards it to CC, (3) CC performs virtual network set-up (hardware and IP addresses, and firewall rules) and schedules the request to a NC, (4) NC retrieves image files from Walrus (or cache), (5) NC starts the VM through the hypervisor, and (6) the user logs into the VM.

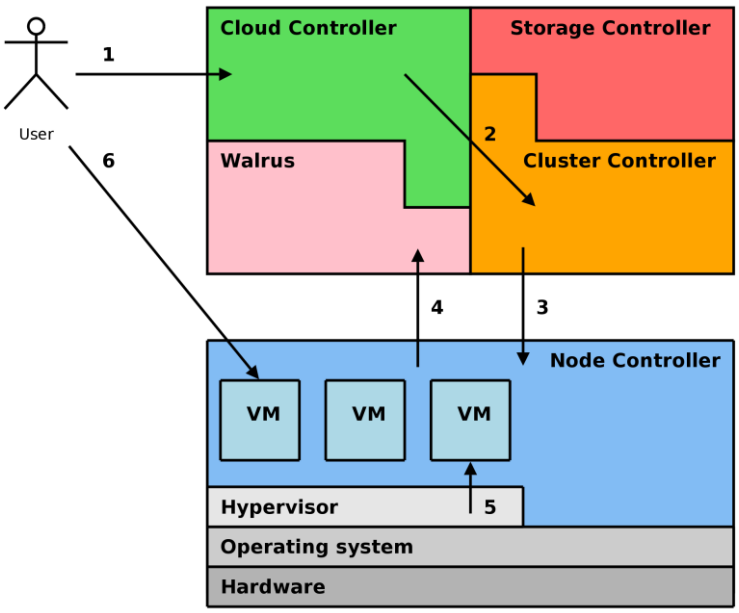

Figure 2. VM deployment with ECC.

To test a medium-sized ECC standard installation we have used the cloud resources available from the FutureGrid project [17], in particular those in the “India” cloud.

One of the first problems we encountered using FutureGrid ECC was the identification of a kernel/ramdisk combination, among those already deployed, working with our own Linux root filesystem image. Using a kernel which is not the one bundled (and tested) with the chosen Linux distribution may cause problems, in addition to the fact that we had to find out and install the corresponding kernel modules.

Another problem that we had to face was the mismatch between the devices referenced by our own Linux root filesystem image, and the para-vitualized devices seen by the Xen kernel.

A simple measure of the time needed to prepare the root filesystem image is the time interval that occurs between instance submission and the VM entering the running state. For a root filesystem image of 4 GB (741 MB compressed) we have measured a start-up time of approximately 5-7 minutes.

4. Enhancing Eucalyptus

In this section we describe the experiments we have performed and the enhancements we have tested starting from a clean ECC v2.0.3 installation. Such enhancements involve, in addition to the tuning of standard configuration files, both customization/modification of ECC scripts and the application of a couple of simple patches to the source code; the scripts and the patches are available to interested readers [18].

4.1. Service environment

As a preliminary step to our experimentation, we have deployed a new ECC installation starting from sources, where all services, except NC, run on VMs so as to optimize hardware resource usage. The answer to the obvious question, "is VM performance capable of efficiently running ECC services?", is positive at least in the following environment that we have tested:

- VMs are executed on a host supporting hardware-assisted virtualization by using a recent version of the open-source machine emulator and virtualizer QEMU [19] combined with Kernel-based Virtual Machine [20] (KVM);
- VM virtual disks are stored in physical disk partitions as logical volumes (LVM) [21] (LVM allows live backups without service interruption);
- storage is made available to both Walrus and SC by an efficient network filesystem (IBM General Parallel File System [22] in our case) to provide the necessary disk space and performance (we plan to test the Lustre [23] filesystem, too); and
- VMs use the *virtio* [24] network and disk drivers.

4.2. Hypervisor

The next step has been to replace the Xen hypervisor with QEMU/KVM on node controllers, considered that all recent servers support hardware-assisted virtualization. The replacement is simple since ECC interfaces with the libvirt API which supports both Xen and QEMU/KVM.

QEMU/KVM on node controllers is configured to use virtio network/disk drivers. We have performed a set of I/O tests: the combination of KVM on host and virtio on guests, except for some unhappy occurrence of host kernel and KVM modules (e.g. hosts running CentOS 5.7 Linux), guarantees a good guest performance (e.g. latest CentOS 6 and Debian Squeeze Linux, but also Windows XP). ECC over QEMU/KVM adopts a different strategy for VM filesystem layout: the image of the root filesystem partition is converted into a virtual disk image in "raw" format immediately before starting a VM instance and the caching is performed on the virtual disk.

One of the advantages of QEMU/KVM is the usage of standard unmodified operating system kernels both in hosts (only kernel modules are required) and guests (Linux and Windows), thus simplifying host and guest set-up and portability.

In addition, QEMU/KVM allows node controllers to boot a VM using the operating system boot loader (e.g. Linux GRand Unified Bootloader or GRUB) stored in the master boot record of the virtual disk image, and hence to boot the operating system from the native kernel/ramdisk stored inside the root filesystem. In this way users can simply use their own kernels/ramdisks and can update them as needed. Of course, this implies the possibility of uploading an entire virtual disk image to Walrus, and not only the root filesystem image, and ECC allows it.

VM start-up time is another aspect that deserves substantial enhancement. The availability of a network filesystem on node controllers allowed us to configure a centralized caching directory shared by all NCs; virtual disk creation from the root filesystem image stored on Walrus is then performed only once for all NCs instead of once for every NC.

4.3. Image format

Unfortunately, a shared cache does not avoid the necessity of copying the virtual disk image from the cache every time a VM is instantiated. In this respect QEMU helped us thanks to the "qcow2" [25] image file format. "Qcow2" is an incremental and differential file format employing the copy-on-write principle of operation; given an initial read-only "raw" image (or backing file), it is possible to create a corresponding read-write incremental "qcow2" virtual disk image pointing to the "raw" backing file. The initial "qcow2" file will be in fact empty (except for some meta-data) and will grow over time only when a write operation is performed on it, while leaving the "raw" backing file unmodified. In other words, the "qcow2" file stores only the differences with the unmodified "raw" backing file. More than one "qcow2" image file can reference the same read-only "raw" backing file, hence the possibility of using the "raw" virtual disk images in the caching directory as backing files for the "qcow2" virtual disk images used by VM, thus avoiding the copy of the whole virtual disk image from cache. The creation of a "qcow2" virtual disk image with a "raw" backing file is very fast (few seconds), and VM start-up times are drastically reduced to a few seconds, if the "raw" image already is in cache (with a shared cache, only the first VM instance of a given type will encounter a cache miss).

4.4. Virtual disk performance

No solution is perfect, and the "qcow2" format introduces a penalty in VM disk write performance. To give some numbers, with a Linux ext3 filesystem we measured the following speeds when writing a file of size 1 GB by means of 1024 write and sync operations of 1 MB each: the guest performs at 33 MB/s for a virtual disk image in "raw" format and 19 MB/s for "qcow2", to be compared with the host performance of 39 MB/s measured when writing directly to the filesystem and 33 MB/s when writing to a "raw" virtual disk image.

"Qcow2" with meta-data pre-allocation performs much better, reaching 32 MB/s, but the use of a backing file and meta-data pre-allocation cannot be combined in the current release of QEMU/KVM; the good news is that they will be in the next future.

Table 1 summarizes the results of some write speed measures performed on a CentOS 6 node controller (PM) and a Debian Squeeze guest (VM) running on top of it. It may be interesting to notice that VM performance substantially increases when over-writing the 1 GB file previously written (suggesting that allocation of new disk space is one of the factors that limits the performance of virtual disk images): approximately 40 MB/s for both "raw" and "qcow2" images and aligned with native host performance.

An important point to consider when comparing results is the decrease in write speed that physical disks exhibit when moving from outer to inner cylinders.

Test description	PM (MB/s)	VM (MB/s)
ext3 on phys. dev. (sda3 local)	39	-
ext3 on phys. dev. (sda4 local)	30	29
ext3 on "raw" image (sda3 local)	33	33
ext3 on LVM "raw" image (iSCSI)	25	24
ext3 on LVM "raw" image (overwrite)	16	16
ext3 on "raw" image (overwrite)	-	41
ext3 on "qcow2" image (sda3 local)	-	19
ext3 on "qcow2" image (overwrite)	-	42
ext3 on "qcow2" preall. image (sda3 local)	-	32
ext3 on "qcow2" preall. image (overwrite)	-	42

Table 1. Summary of write speed tests.

To better understand the data shown in **Table 1**, it may be useful to know that sda4 is a 10 GB partition that spans only over the inner disk cylinders, while sda3 is a 180 GB partition (almost empty) whose extension starts from the outer cylinders: this leads to different write speeds, 30 MB/s for sda4 and 39 MB/s for sda3, the outer the better.

The "qcow2" performance problem is more annoying for Windows guests and especially immediately after boot, since this operating system uses a page file located inside its root filesystem; during operating system start-up all process data that are not in active use are written to the page file causing a lot of (slow) writes to the "qcow2" image.

At a first sight "qcow2" performance could be a relevant problem, since in general disk I/O performance for VM is not exceptional (at least when the virtual disk images are stored into files as with ECC). However, we must consider that heavy disk write activity on the VM root filesystem is not recommended anyway, nor it is usually necessary since ECC provides off-instance volumes that can be attached on-the-fly to a running VM instance and be seen as standard disks. Off-instance volumes are "exported" by a storage controller and "imported" by the host running the VM via the iSCSI protocol, i.e. the network, and attached to the VM by emulating a PCI hot-plug device. If the storage controller features a good filesystem performance and the network bandwidth is reasonable, volume performance is acceptable: the 1 GB test executed on an attached volume reached 24 MB/s on the guest, to be compared with 25 MB/s on the host. If the storage controller can be attached to a high performance NAS or SAN, a higher speed would be achieved.

4.5. I/O barriers

Filesystem performance on Linux is influenced by many factors such as hardware, filesystem type and mount options. In particular, I/O barriers [26] on journaling filesystems (i.e. ext3 and ext4) may substantially slow down sync operations. This point is somehow subtle since different Linux distributions implicitly enforce different default mount options; as an example, going from CentOS 5.7 to CentOS 6.0, the barrier default setting changes from *off* to *on* with a considerable performance slow-down for our 1 GB write test (16 MB/s with barriers on and 48 MB/s with barriers off). To recover the previous behavior we had to explicitly set *barrier=0* in CentOS 6.0 mount options.

4.6. File injection

Related to the "qcow2" image format, is the problem of "injecting" files into a virtual disk image before VM start-up; this technique is used by ECC to customize the VM with a user-generated public key for the root user, in such a way that after boot the user can exploit secure shell to log in to the VM, by using the corresponding private key (Linux only). While the injection of a file into a "raw" image can be performed directly by using the *loopback* devices, the same is not true for a "qcow2" image. In the latter case we have then resorted to the *libguestfs* [27] suite.

5. Porting new machines into Eucalyptus

In our experience, when a user wishes to run guests from its own deployed root filesystem or disk image, he/she may immediately run into troubles, due to the differences in the configuration expected by ECC and the one exhibited by a cloned PM or even VM built for a different environment. In this section we present a to-do list that may hopefully help in the aforementioned task of configuring and running a user-provided Linux guest in ECC. Please notice that not all ECC installations have the same configuration, as the experiments reported in this work should point out. At a minimum, users should be aware of the hypervisor in use, i.e. Xen or QEMU/KVM.

5.1. Filesystem layout

In many recent Linux distributions (such as RHEL/CentOS or Debian/Ubuntu), disk partitions are no longer referred to by device in */etc/fstab*; labels or universally unique identifiers (UUID) are preferred just because of physical device independence. Unluckily, some cloning tools (most noticeably ECC euca-bundle-vol) do not clone labels or UUID; hence, if */etc/fstab* is not manually adjusted in the cloned image, a mismatch occurs and partitions cannot be mounted during VM boot. A similar problem can come about even if */etc/fstab* refers to disk partitions by device, since devices usually differ between a running PM and its VM clone or between a running Xen VM and its QEMU/KVM clone; the SATA hard disk partition device */dev/sda1* in a PM may become the virtio disk partition */dev/vda1* in a QEMU/KVM VM or the virtual disk partition */dev/xvda1* in a Xen VM.

5.2. Network

Network configuration is simple, since ECC provides the IP address to its guests via the DHCP protocol (on the contrary, host network configuration is quite flexible [3]): just set the network interface *eth0* to acquire an IP address via DHCP; a common error, especially when cloning a PM, is to have a specific MAC or hardware address assigned to the network interface: this must be removed from the network configuration file because ECC assigns its own MAC addresses to guests and a MAC mismatch will cause network start-up to fail.

5.3. Serial console

ECC can redirect the output of the VM serial console into a file, which can then be viewed by the user; this may be useful for debugging problems which occur during VM start-up. For console redirection to work, a VM must be configured to write to its serial console, usually by enabling the device */dev/ttyS0* (KVM) or */dev/hvc0* (Xen) in the */etc/inittab* configuration file. No serial console configured means no output redirected to the file.

5.4. Ramdisk

The ramdisk associated to a kernel image contains the modules necessary to mount the root filesystem during boot; when booting a VM inside ECC, additional modules may be needed, notably the virtio modules if QEMU/KVM is configured to reply upon them. In such a case, the ramdisk must be re-created including the necessary modules; otherwise the boot will fail when trying to mount the root filesystem. A common error is to clone a PM or a VM built for Xen expecting that it will boot as a VM using QEMU/KVM configured to employ virtio: a typical PM ramdisk does not contain virtio device drivers.

5.5. Boot loader

Booting a VM requires to specify at least kernel, ramdisk and root filesystem. These boot parameters can be provided off-image by the hypervisor or can be configured into a boot loader such as GRUB, installed on the virtual disk image. Kernel and ramdisk deserve no special care, apart from being compatible with the VM image (do not forget the kernel modules), and users can choose their own configuration both in the off-image and on-image case. On the contrary, the root filesystem parameter requires special attention. While in the on-image case users can configure such parameter by configuring GRUB, in the off-image case it is out of user control, because it is statically configured. Anyway, if there is a mismatch between the root filesystem parameter and the actual location of the root filesystem inside the virtual disk image, VM boot will fail. Hence: (1) if the root filesystem is specified by label, then the corresponding partition must have the same label, or (2) if the root filesystem is specified by UUID, then the same UUID must be present on the partition, or (3) if the root filesystem is specified by device, then such a device

must exist in the VM. QEMU/KVM configured for virtio will not boot a VM if the root filesystem is specified as */dev/hda1* since the typical device for such a VM is */dev/vda1*; on the contrary, QEMU/KVM configured for IDE disk emulation will boot a VM if the root filesystem is specified as */dev/hda1*, not if it is specified as */dev/vda1*.

5.6. Windows

Windows is not covered in detail here, but unless QEMU/KVM or Xen/HVM are configured to emulate devices known to Windows (e.g. IDE disks), the correct device drivers must be installed. While filesystem layout, ramdisk and boot loader should not require special configuration, almost certainly a Windows re-activation will be required if using a cloned disk image. As far as the network is concerned, DHCP should be configured and terminal services started to be able to log in to the VM through the network.

6. Conclusion

Eucalyptus Community Cloud is a very interesting cloud-computing framework with promising possibilities, relying on open-source code and on EC2 and S3 API, for which a public specification is available. In this work we have investigated and proposed a number of enhancements and extensions that, in our belief, can make ECC even more attractive for building both private and community cloud infrastructures, but also with an eye toward public clouds.

Many other interesting features can be tested, among others we plan to experiment with the new "qcow2" format supporting both backing file and meta-data pre-allocation, as soon as it is available.

In addition, most of the attracting features that made us prefer KVM over Xen seem to be available in the latest Xen, i.e. version 4, and we plan to investigate the combination of ECC and Xen4 in the next future, even because every Linux kernel from 2.6.39 onwards will contain the Xen hypervisor, thus eliminating the need of a modified kernel in hosts and guests.

Moreover, some of the points covered in Section 5 could be automatically managed within ECC by means of suitable scripts; for example, we already have in place a semi-automatic procedure for installing GRUB to a virtual disk image, and we plan to include it in ECC as a completely automatic operation.

7. Acknowledgements

The authors acknowledge the Cybersar Consortium for the use of its computing facilities. This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812.

8. References

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing", Special Publication 800-145, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2011.

[2] Open Cloud Manifesto, 2009. http://www.opencloudmanifesto.org/Open%20Cloud%20Manifesto.pdf

[3] D. Nurmi *et al.*, "The Eucalyptus Open-Source Cloud-Computing System", *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, Shanghai, 2009, pp. 124-131. doi:10.1109/CCGRID.2009.93

[4] Amazon Elastic Compute Cloud API Reference, 2011. http://awsdocs.s3.amazonaws.com/EC2/latest/ec2-api.pdf

[5] Amazon Simple Storage Service API Reference, 2006. http://awsdocs.s3.amazonaws.com/S3/latest/s3-api.pdf

[6] J. E. Smith, R. Nair, "The Architecture of Virtual Machines", *Computer (IEEE)*, Vol. 38, No. 5, 2005, pp. 32–38. doi:10.1109/MC.2005.173

[7] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, “Service-Oriented Computing: State of the Art and Research Challenges”, *Computer (IEEE)*, Vol. 40, No. 11, 2007, pp. 64-71. doi:10.1109/MC.2007.400

[8] Euca2ools, 2011.
http://open.eucalyptus.com/wiki/Euca2oolsGuide_v1.3

[9] M. Ludvig, “S3 tools”, 2011.
http://s3tools.org/s3tools

[10] Hybridfox, 2011.
http://code.google.com/p/hybridfox

[11] Xen, 2011.
http://xen.org

[12] libvirt: The virtualization API, 2011.
http://libvirt.org

[13] T. Abels, P. Dhawan, B. Chandrasekaran, “An Overview of Xen Virtualization”, *Dell Power Solutions*, August 2005, pp. 109-111.

[14] AMD64 Virtualization Codenamed “Pacifica” Technology - Secure Virtual Machine Architecture Reference Manual, Advanced Micro Devices, May 2005.

[15] G. Neiger, A. Santoni, F. Leung, D. Rodgers, R. Uhlig, “Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization”, *Intel Technology Journal*, Vol. 10, No. 3, 2006, pp. 167–178. doi:10.1535/itj.1003.01

[16] A. Aneja, “Xen Hypervisor Case Study – Designing Embedded Virtualized Intel Architecture Platforms”, Intel, March 2011.

[17] FutureGrid: a distributed testbed for Clouds, Grids, and HPC, 2011
https://portal.futuregrid.org

[18] A. Bosin, M. Dessalvi, G. M. Mereu, G. Serra, “Enhancing Eucalyptus Community Cloud”, 2011.
http://www.dsf.unica.it/~andrea/eucalyptus.html

[19] QEMU, 2011.
http://wiki.qemu.org/Main_Page

[20] Kernel Based Virtual Machine, 2011.
http://www.linux-kvm.org/page/Main_Page

[21] LVM, 2011.
http://sources.redhat.com/lvm2

[22] IBM General Parallel File System, 2011.
http://www-03.ibm.com/systems/software/gpfs

[23] Lustre filesystem, 2011
http://wiki.lustre.org/index.php/Main_Page

[24] R. Russell, “virtio: towards a de-facto standard for virtual I/O devices”, *ACM SIGOPS Operating Systems Review - Research and developments in the Linux kernel archive*, Vol. 42, No. 5, 2008, pp. 95-103. doi:10.1145/1400097.1400108

[25] M. McLoughlin, “The Qcow2 image format”, 2008.
http://people.gnome.org/~markmc/qcow-image-format.html

[26] Understanding Linux block IO barriers, 2010.
http://www.linuxsmiths.com/blog/?p=18.

[27] libguestfs: tools for accessing and modifying virtual machine disk images, 2011.
http://libguestfs.org

RINGRAZIAMENTI

Vorrei ringraziare Simona e la mia famiglia per il loro sostegno sempre presente. Un grazie speciale a prof. Andrea Bosin sempre disponibile e partecipe con preziosi suggerimenti e a prof. Gianni Fenu grazie al quale questa avventura ebbe inizio.

Grazie inoltre a tutti coloro che in qualche modo hanno contribuito alla stesura di questa tesi tra cui Matteo Dessalvi, i colleghi di studio e di lavoro.