Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali
Corso di Laurea Triennale in Informatica

Progettazione e sviluppo di un algoritmo decentralizzato per il consolidamento di Macchine Virtuali

Tesi di Laurea in Algoritmi e Strutture Dati

Relatore:
Chiar.mo Prof.
Moreno Marzolla

Presentata da:
Lorenzo Severini

Sessione II
Anno Accademico 2010 - 11

A Roberto, Antonella e Silvia

Introduzione

Il Cloud Computing è un paradigma che, oggigiorno, è sempre più usato per offrire servizi di vario genere via web.
Questo modello permette l'accesso e l'utilizzo via rete in maniera condivisa, pratica e on-demand di risorse di calcolo (es reti, server, memorie di archiviazione, servizi, applicazioni) accessibili e rilasciabili in maniera rapida e con la minima interazione con il fornitore [16].
Il fornitore del servizio necessita di un data center con un numero elevato di calcolatori in grado di soddisfare le richieste di tutti gli utenti; per ogni servizio viene solitamente creata una macchina virtuale in grado di interagire remotamente.
Il costo maggiore di cui il gestore della Cloud deve farsi carico è quello dovuto al consumo energetico dei server, difficilmente gestibile in quanto le richieste presentano una grande variabilità di carico.
Lo scopo di questa Tesi è quindi quello di fornire un algoritmo per il consolidamento delle macchine virtuali nei server in modo da massimizzare l'utilizzo degli elaboratori potendo così spegnere quelli in cui non è in esecuzione nessuna macchina virtuale con un conseguente risparmio energetico.
Nel primo capitolo verranno enunciate le caratteristiche principali delle Macchine Virtuali, del Cloud Computing e, più in generale, delle tecniche di risparmio energetico nei data center.
Nel secondo capitolo verranno descritti i due algoritmi distribuiti di Gossip che compongo il software realizzato: l'algoritmo V-MAN che si occupa del consolidamento delle Macchine Virtuali e l'algoritmo Peer Sampling il cui

obbiettivo è di fare in modo che ogni nodo possieda una conoscenza della rete sempre aggiornata.

Nel terzo capitolo, verranno valutate le performance dell'algoritmo effettuando vari test in diverse condizioni.

Infine, nella conclusione, vengono suggeriti alcuni sviluppi futuri per migliorare le funzionalità e le caratteristiche del software.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Contesto Scientifico

1.1 Le Macchine Virtuali

1.1.1 La Virtualizzazione

La virtualizzazione può essere definita formalmente come un isomorfismo tra un sistema ospitante (reale) e un sistema ospitato (virtuale) [17].
È realizzata inserendo un livello intermedio in grado emulare le risorse computazionali (hardware e software)che fornisce un'interfaccia omogenea ai livelli superiori.
Lo scopo è quello di superare i vincoli di una macchina fisica (architettura, periferiche, sistema operativo) in modo da incrementarne la flessibilità, fornendo ai livelli superiori una visione astratta del sistema.
È bene sottolineare la differenza tra emulazione, cioè la riproduzione delle funzionalità, e simulazione, cioè la riproduzione del comportamento visibile.
Il software usato per questo fine viene chiamato **Macchina Virtuale** (VM).
Storicamente [10] la prima VM è stata introdotta negli anni 60 da IBM con una tecnologia chiamata VM/370: la sua interfaccia era la stessa del sistema host e aveva lo scopo di creare un ambiente protetto per ogni singolo utente.
Più tardi, negli anni 90, con l'obiettivo di creare un linguaggio portabile, venne creata la Java Virtual Machine (JVM) la quale interpreta il byteco-

de (un linguaggio intermedio in cui viene tradotto il codice scritto in Java), rendendo così possibile l'esecuzione del programma su qualsiasi tipo di architettura.
Oggi questi software sono usati principalmente per sviluppo e test di programmi di vario tipo, per retrocompatibilità con architetture diverse da quelle dell'host o per l'isolamento degli utenti.

1.1.2 Classificazione

Le macchine virtuali possono essere suddivise in due macrocategorie a seconda della granularità dell'enità virtualizzata. Nella figura 1.1 viene rappresentata la differente posizione del livello di virtualizzazione inserito all'interno di un elaboratore.

- **Process Virtual Machine**:
  il software di virtualizzazione si posizione al livello sottostante ad un singolo processo (tra processo e Sistema Operativo).

- **System Virtual Machine**:
  il software di virtualizzazione emula un ambiente di sistema appoggiato immediatamente sopra il livello hardware.

1.1.3 Software per la virtualizzazione

Esistono un gran numero di software per la virtualizzazione: qui di seguito ecco un elenco dei più famosi .

- **VM eterogenee/emulatori**:

  Qemu opensource, eseguibile in Linux, è in grado di emulare diverse architetture. Fornisce un ambiente completo e permette la conversione tra diversi Instruction Set Architecture (ISA). Consente anche di eseguire programmi compilati su altre architetture.

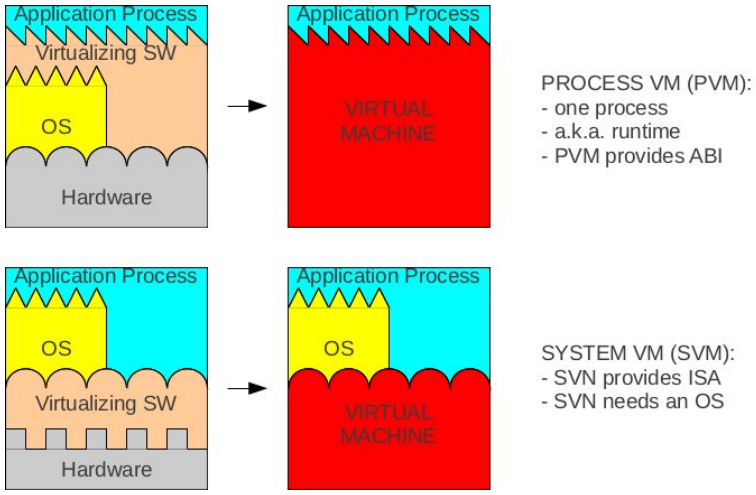

Figura 1.1: Schema di PVM e SVM [9]

Inoltre possiede un modulo kernel chiamato *kqemu* usato, quando possibile (es. architettura host uguale a quella emulata), per migliorare le prestazioni.

PearPC emula l'architettura PowerPC. Fornisce un supporto completo dell'hardware anche se con un forte decremento delle prestazioni.

MacOnLinux permette di eseguire MacOS in un sistema Linux con architettura PowerPC utilizzando un modulo kernel (e privilegi di root) con buone prestazioni.

- **VM omogenee/hypervisors**:

  KVM è l'acronimo di Kernel Based VM, sfrutta le tecnologie hardware Intel VT e AMD-V ed è incluso nel kernel di Linux. È un monitor di VM e supporta anche la paravirtualizzazione [1]

  VirtualBox è prodotto dalla Oracle e fornisce una virtualizzazione su architettura x86 permettendo di emulare diversi Sistemi Operati-

[1] **paravirtualizzazione**: non consiste nell'emulazione dell'hardware ma nel controllo all'accesso delle risorse.

vi (MacOS, Windows, Linux, Solaris). È disponibile in versione opensource e proprietaria.

Xen è un progetto IBM .Utilizzando la paravirtualizzazione e permette la una virtualizzazione in modo efficiente. Per il supporto hardware si affida ai device driver della prima VM attivata (domain 0)

- **VM a livello SO** :

  Virtuozzo possiede il concetto di VPS (Virtual Private Server): una collezione di applicazioni che condividono gli stessi ambienti virtuali. Non fornisce l'esecuzione di più kernel contemporaneamente: ogni VPS utilizza quello in esecuzione sulla macchina host.

- **VM a livello processo** :

  User-Mode Linux permette di eseguire più sistemi Linux in user space come fossero processi: ogni system call eseguita sui kernel virtuali viene trasmessa al kernel della macchina ospite che la esegue realmente.

1.1.4 La Migrazione live

Per **Migrazione live** [8] si intende il trasferimento di una VM da una macchina fisica verso un'altra. Viene detta live in quanto questa operazione viene fatta durante l'esecuzione della VM stessa. Le informazioni fondamentali da trasferire sono:

- Stato CPU
- Contenuto della RAM
- Contenuto delle memorie fisiche (storage)
- Connessioni di rete

KVM, Xen, VMware sono un esempio di software che implementano questa funzione.
Esempio utilizzando Xen:

```
# xm migrate -live nome_VM host_destinazione
```

1.2 Il Cloud Computing

1.2.1 Definizione

Con il continuo sviluppo delle tecnologie in ambiente Internet che permettono un'ampia interazione tra sito e utente e che sono racchiuse nel termine Web 2.0, è diventato di primaria importanza un paradigma che prende il nome di Cloud Computing.
Come definito da [6] una **Cloud** è un tipo di sistema parallelo e distribuito formato da una collezione di elaboratori interconnessi e virtualizzati, presentati come una o più risorse di calcolo unificate.
Il National Institute of Standard and Technology (NIST) definisce il **Cloud Computing** [16] come un modello che permette l'accesso e l'utilizzo via rete in maniera condivisa, pratica e on-demand di risorse di calcolo (es reti, server, memorie di archiviazione, servizi, applicazioni) accessibili e rilasciabili in maniera rapida e con la minima interazione con il fornitore.
Le principali caratteristiche di questa tecnologia sono [12]:

scalabilità: è una proprietà fondamentale di una cloud e descrive la capacità dell'infrastruttura di adattarsi ai cambiamenti come, ad esempio, la quantità e la dimensione dei dati usati da un'applicazione e il numero di utenti connessi contemporaneamente.
Possiamo distinguere tra scalabilità orizzontale, che riguarda il numero di richieste da soddisfare e scalabilità verticale, che riguarda la quantità di risorse utilizzate per soddisfare le richieste.
In una Cloud è necessaria la presenza sia di una scalabilità verso l'alto (up-scaling), che di una scalabilità verso il basso (down-scaling).

affidabilità: è una caratteristica essenziale di una Cloud e descrive la capacità di offrire servizi senza interruzioni e malfunzionamenti. Generalmente è attuata mediante l'utilizzo di risorse ridondanti.

disponibilità continua di dati e servizi. Consiste nel'introdurre un'adeguata ridondanza in modo che, in caso di guasti improvvisi, questi siano mascherati in maniera trasparente per l'utente, senza nessun decremento significativo delle performance.

pay per use. La capacità di stabilire il costo in base all'effettivo utilizzo delle risorse è una caratteristica peculiare di un sistema Cloud. Questa proprietà è fortemente legata alla qualità del servizio (QoS) in quanto i particolari requisiti richiesti sono pagati di conseguenza.

1.2.2 Tipi di Cloud

In questa sezione vengono classificati i vari modelli di Cloud in base alla tipologia dei servizi offerti. Nella figura 1.2 sono rappresentate le posizioni delle varie categorie rispetto al livello hardware della macchina.

SaaS (Software as a Service) [3, 16]: questo livello fornisce la possibilità di usufruire delle applicazione eseguita sulla Cloud attraverso un client (es. un browser web). (es. Google Docs, Amitive Unity,. . .)

PaaS (Platform as a Service): questo livello fornisce all'utente, pur non avendo il controllo dell'intera infrastruttura, una piattaforma in cui può distribuire proprie applicazioni eseguite poi remotamente. (es. Windows Azure, Google Apps,. . .)

Iaas (Infrastructure as a Service): questo livello fornisce all'utente il controllo delle componenti fondamentali di un sistema di calcolo come la rete, la CPU e le memorie di archiviazione con la possibiltà di eseguire proprie applicazioni. Sebbene non controlli l'interna cloud può gestirne il Sistema Operativo. (es. Open Nebula, Eucalyptus, Amazon EC2,. . .).

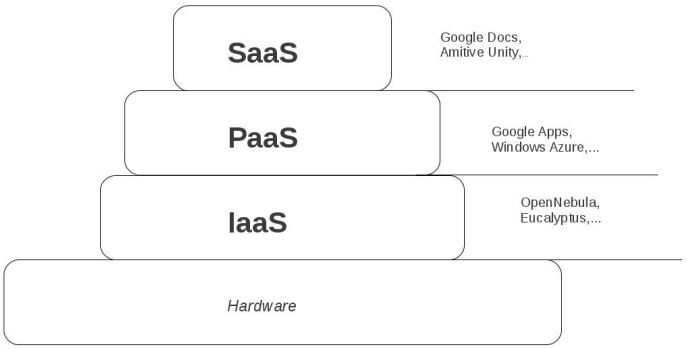

Figura 1.2: Schema delle varie tipologie di Cloud Computing con esempi.

1.2.3 Deployment Model

In questa sezione vengono descritti i vari modelli di infrastruttura della Cloud in base al modo di utilizzo e di accesso ai dati.

Private Cloud:[2] l'infrastruttura della Cloud è accessibile solo all'interno di una determinata organizzazione. É solitamente gestita dall'organizzazione stessa o da terze parti.

Community Cloud: l'infrastruttura della Cloud è condivisa da una o più organizzazioni aventi un obiettivo comune.

Public Cloud : l'infrastruttura della Cloud è generalmente creata da grandi industrie o da enti pubbliche e, essendo accessibile esternamente, fornisce servizi ad ogni utente in maniera pay-as-you-go.

Hybrid Cloud: l'infrastruttura della Cloud è una composizione di più modelli di Cloud. Possono essere presenti nello stesso tempo parti accessibili solo internamente (Private Cloud) o pubblicamente (Public Cloud).

1.2.4 Problemi strutturali

Il Cloud Computing attualmente soffre di alcuni problemi strutturali che ne impediscono una crescita ulteriore nell'utilizzo.

Disponibilità di servizio: è il problema più significativo in quanto un eventuale guasto o malfunzionamento del Cloud provider rende inutilizzabile qualunque servizio, provocando un danno economico di importante entità. Questo svantaggio può anche essere sfruttato a scopo estorsivo da criminali utilizzando l'attacco DDoS (Distribuited Denial of Service).

Riuso dei dati: le API utilizzate nella Cloud, non essendoci ancora una standardizzazione, sono essenzialmente proprietarie e di conseguenza può essere difficile estrarre i propri dati e programmi per eseguirli su un'altra piattaforma.

La mancanza di uno standard aperto e riconosciuto universalmente causa quindi una vera e propria dipendenza dell'utente che può essere soggetto, ad esempio, ad un aumento incontrollato del costo del servizio senza possibilità di migrare verso un provider più conveniente.

Riservatezza dati: essendo i dati conservati in server remoti, è possibile, in mancanza di una qualche forma di crittografia, che questi possano essere usati da terzi per scopi più o meno leciti. Inoltre può succedere che questi siano conservati in data center in stati con una legislazione non adeguata in materia di privacy.

Velocità trasferimento dati: se al giorno d'oggi non vi sono problemi nell'utilizzare una grande quantità di dati per le applicazioni eseguite in locale (sono disponibile dischi di qualche TeraByte a poche centinaia di dollari), diverso è il discorso se si utilizza la rete.

Le comuni infrastrutture hanno una capacità inadatta per il trasferimento di grandi quantità di dati. Da sottolineare che, inoltre, possono essere presenti dei costi per il trasferimento di dati. Sarebbe necessa-

rio quindi l'utilizzo di protocolli di trasferimento che minimizzino la quantità di informazioni scambiate.

1.3 Il consumo energetico nei Data Center

1.3.1 Il problema

Con l'avvento del Cloud Computing, di cui abbiamo parlato nella sezione precedente, ha sempre più rilevanza il problema dell'elevato consumo energetico dei Data Center, le unità organizzative che mantengono le apparecchiature ed i servizi di gestione dati.
È stato calcolato [5] che gli USA, nel 2006, sfruttavano tra l'1,5 e il 3% dell'elettricità totale.
Inoltre, applicando la nota Legge di Moore, questo dato (e di conseguenza la quantità di denaro speso) avrà, nei prossimi anni, una crescita quasi esponenziale. Questo dato è dovuto, oltre all'alimentazione dell'hardware, anche agli impianti di raffreddamento il cui impatto sulla quantità di energia totale impiegata é stimato tra il 60 e il 70% [4] (infatti oggi c'è la tendenza a spostare i Data Center in paesi con una temperatura media molto bassa).

1.3.2 Le soluzioni

Vi è una continua rincorsa per cercare di risolvere il problema dell'elevato consumo energetico dei Data center.
Come descritto in [4] un metodo è senz'altro l'utilizzo di hardware efficienti dal punto di vista dei consumi. Un esempio sono i moderni dischi a stato solido: necessitano di una minore quantità di energia rispetto ai classici hard disk.
È anche possibile ridurre il consumo di un elaboratore limitando il consumo dei vari ocmponenti. Sono presenti diverse tecnologie che permettono di diminuire la potenza del processore: per fare ciò diminuiscono la velocità

del clock della CPU (clock gating) oppure spengono temporaneamente alcuni componenti se inattivi (idle).
È stato introdotto anche uno standard chiamato ACPI (Advanced Configuration and Power Interface) che definisce una serie di stati per la gestione energetica globale (da G1 a G3) e delle periferiche (da D0 a D3)[1].
A livello software, invece, è stata studiata la possibilità di uno scheduler Real-Time multiprocessore il quale, considerando probabilisticamente la distribuzione dei vari task, li esegue riducendo il consumo [7].

1.3.3 Il consolidamento server nel Cloud Computing

La minimizzazione dell'utilizzo di energia interessa chiaramente anche le facility che forniscono un servizio di Cloud Computing.
Questo paradigma infatti, oltre a soffrire dei problemi descritti in precedenza, ha, per costruzione, la questione dell'enorme variabilità di carico dipendente dal numero e dal peso delle operazioni richieste dall'utente.
Un'approccio molto funzionale alla risoluzione del problema è il consolidamento dei server attraverso l'uso di Macchine Virtuali, usate per fornire i servizi richiesti.
Per consolidamento si intende la riduzione del numero degli elaboratori attivi in un centro di elaborazione.
Ciò avviene lasciando invariati i servizi offerti e ottenendo una diminuzione del consumo energetico.
Questo processo risolve il problema del server sprawl, ovvero la presenza in un data center di un numero di server attivi ma con una scarsa percentuale di utilizzo.
Il vantaggio nell'utilizzo della macchine virtuali sta nel fatto che queste possono essere migrate (vedi sez 1.1.4) da un server all'altro in modo da massimizzarne l'utilizzo.
I server svuotati di ogni VM possono essere così spenti o messi in modalità a basso consumo.
È proprio questo lo scopo dell'algoritmo descritto nel prossimo capitolo.

Capitolo 2

Progettazione e Implementazione

Lo scopo di questa Tesi è l'implementazione di un algoritmo per il consolidamento delle macchine virtuali nei server in modo da massimizzare l'utilizzo degli elaboratori potendo così spegnere quelli in cui non è in esecuzione nessuna macchina virtuale con un conseguente risparmio energetico.
L'algoritmo è completamente distribuito e basato su un protocollo di Gossip senza coordinamento centrale. In un algoritmo di questo tipo [11], come suggerisce il nome, le informazioni si propagano con una sorta di passaparola, come appunto i gossip e i rumors fra le persone comuni. Per il modo in cui si propaga l'informazione viene anche detto epidemico. È quindi completamente asincrono e decentralizzato.
Questo protocollo viene utilizzato nelle reti che interconnettono nodi con la possibilità di comunicare tra di loro: ogni nodo scambia i propri dati solamente con un sottoinsieme finito di nodi scelti casualmente.
L'algoritmo di consolidamento dei server implementato e descritto in questo elaborato è essenzialmente formato da due parti principali: la prima si basa sull'algoritmo V-MAN descritto in [15] che si occupa della gestione delle macchine virtuali, la seconda si basa sull'algoritmo di Peer Sampling descritto in [14] e si occupa della creazione e del mantenimento del grafo dei nodi.

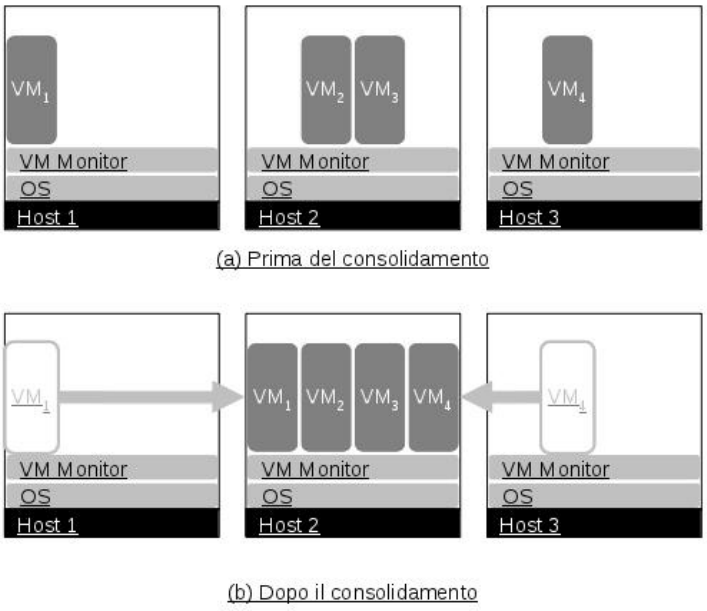

Figura 2.1: Consolidamento di VM [15]

2.1 V-MAN

V-MAN è un algoritmo epidemico completamente distribuito per il consolidamento di Macchine Virtuali in ambiente Cloud Computing.
È basato su un semplice protocollo di Gossip senza coordinamento centrale.
Viene eseguito periodicamente: una volta avvenuto il consolidamento si passa alla migrazione vera e propria.
Nella figura 2.2 è schematizzata la procedura di consolidamento server.

2.1.1 Descrizione formale del problema

Consideriamo N server identificati con un numero $1..N$.
Supponiamo di avere una Cloud in cui ogni server può scambiare messaggi con gli altri.
Ogni server può entrare ed uscire dalla Cloud in qualsiasi momento.
Indichiamo con C il numero massimo di VM eseguibili su un singolo server.

$H = (H_1, \ldots, H_N)$ è il vettore di allocazione delle VM dove $H_i \in \{0, \ldots, C\}$ e rappresenta il numero di VM in esecuzione sul server i.
Rappresentiamo con M la somma di tutte le VM in esecuzione (ovvero la somma di tutti gli elementi del vettore H).
Con $F_k(\mathbf{H}) = |\{i : H_i = k\}|/N$ con $k = 0, \ldots, C$ indichiamo la frazione di host che contengono esattamente k VM.
Nella tabella 2.1 sono elencati i simboli usati nella descrizione del problema.

Lo scopo dell'algoritmo è quello di massimizzare $F_0(H)$

Gli obiettivi del da raggiungere sono:

auto-organizzazione: l'algoritmo deve operare adeguatamente, senza l'intervento manuale, anche con la presenza di nodi che lasciano o si inseriscono nella rete.

efficienza: l'algoritmo deve produrre una buona soluzione al problema nel tempo più breve possibile.

scalabilità: l'algoritmo deve essere efficiente anche se applicato in una vasta Cloud.

robustezza: l'algoritmo deve essere resistente anche ad un numero elevato di guasti nei server (situazione frequente nei grandi data center).

2.1.2 Pseudocodice dell'algoritmo V-MAN

$i \leftarrow$ GetProcId()
procedure ActiveThread
loop
  Wait Δ
  $j \leftarrow$ GetRandPeers(i)
  Send $\langle H_i \rangle$ to j
  Receive $\langle H'_i \rangle$ from j

$H_i \leftarrow H'_i$
end loop
procedure PassiveThread
loop
Wait for message $\langle H_j \rangle$ from j
if $(H_i > H_j)$ **then**
$D \leftarrow \min(H_j, C - H_i)$
Send $\langle H_j - D \rangle$ to j
$H_i \leftarrow H_i + D$
else
$D \leftarrow \min(H_i, C - H_j)$
Send $\langle H_j + D \rangle$ to j
$H_i \leftarrow H_i - D$
end if
end loop

N	Numero di server
C	Massimo numero di VM eseguibili in un server
M	Numero di VM correntemente in esecuzione
H_i	Numero di VM in esecuzione nel server i, $0 \leq H_i \leq C$
$F_k(H)$	Frazione di server con esattamente k VM
$F_{0,opt}(H)$	Frazione ottimale di server vuoti

Tabella 2.1: Tabella dei simboli

2.1.3 Spiegazione e funzionamento dell'algoritmo

Lo pseudocodice descrive V-MAN, un algoritmo di gossip per il consolidamento dei server mediante l'utilizzo di Macchine Virtuali.
V-MAN non richiede nessuna politica particolare per l'allocazione delle VM nei server e nessun procedimento speciale deve essere effettuato alla terminazione delle VM. V-MAN può essere eseguito periodicamente in modo da

compattare le VM in pochi host; inoltre permette l'aggiunta e la rimozione delle VM durante l'esecuzione.
L'algoritmo può essere eseguito in background in modo da non interferire con la normale attività della Cloud.

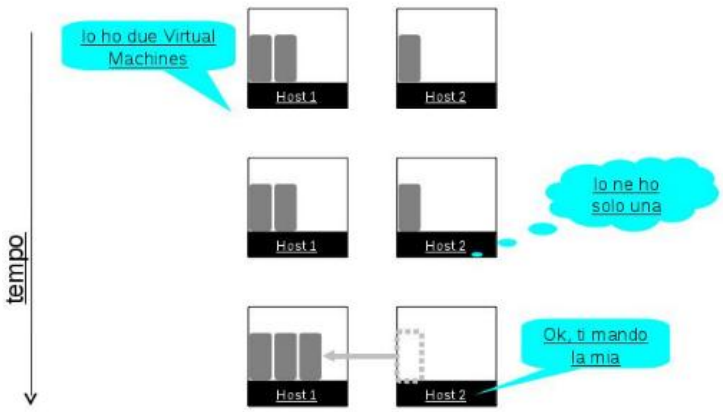

Figura 2.2: Rappresentazione V-MAN

L'allocazione iniziale delle VM è rappresentata dal vettore $H = (H_1, \ldots, H_N)$. Ogni server i è a conoscenza solo del numero H_i VM attualmente ospitate. Inizialmente ad ogni nodo viene assegnato un identificatore. Vengono poi allocate casualmente, rispettando la capacità massima C, le VM presenti in ogni nodo.
Ogni VM è contrassegnata da un codice alfanumerico univoco e dalla coppia indirizzo IP dell'elaboratore su cui è in esecuzione e porta su cui comunica il demone dell'algoritmo V-MAN.
Durante l'esecuzione, ad ogni iterazione, rappresentando la rete con un grafo orientato, ogni nodo comunica il suo numero di VM ad un suo vicino, ovvero un nodo raggiungibile direttamente in un passo (la gestione della rete è

compito del protocollo di Peer Sampling descritto nella sezione successiva). Quello con il numero di VM più alto, a seconda dello spazio a disposizione, riceverà tutte o una parte delle VM dell'altro.
L'algoritmo è composto da due thread: uno attivo (ActiveThread) e un altro passivo (PassiveThread).
L'Active Thread, ogni intervallo Δ, seleziona casualmente un vicino e gli invia il proprio numero di VM (H_i). Subito dopo attende che l'altro host gli restituisca il nuovo numero di VM (H_i) opportunamente aggiornato e lo sostituisce con il proprio. Infine ricomincia, in un loop infinito, le operazioni dall'inizio.
Il Passive Thread invece attende l'arrivo di messaggi da altri vicini. Appena ricevuto il numero di VM in esecuzione su un altro nodo lo confronta con il proprio. Se possiede più VM allora calcola il numero che ne può ospitare (D), invia il numero aggiornato all'altro host e aggiorna il proprio. Viceversa se possiede più VM calcola il numero che può cedere (D) invia il numero aggiornato all'altro host e aggiorna il proprio. Infine ricomincia dall'inizio. Nella figura 2.2 è schematizzato il funzionamento dell'algoritmo mediante la descrizione di una comunicazione tipo tra due nodi.

2.2 Gossip based Peer Sampling

L'algoritmo di Peer Sampling, in una rete formata da un numero variabile di nodi, permette che ogni nodo abbia una conoscenza parziale ma sempre aggiornata della situazione della rete. Ciò avviene mediante lo scambio tra nodi della lista degli ultimi host che hanno manifestato una qualche attività recentemente.
È basato su un protocollo di Gossip ed è completamente decentralizzato. Considerando l'intera rete come un grafo, ogni host è interconnesso con un sottoinsieme finito di peers detti vicini (*neighbours*).

Ogni nodo è descritto in base a un identificatore univoco e a un indirizzo usato per la comunicazione. Inoltre ha una tabella che rappresenta la sua

conoscenza della situazione della rete. Se completa di tutti i nodi viene chiamata vista globale mentre se è composta da un sottoinsieme limitato (come nel nostro caso) viene chiamata vista parziale (c rappresenta la sua dimensione). L'ordine della vista viene modificato solo esplicitamente (permute) e non sono presenti duplicati.
Per ogni host è memorizzato un campo chiamato *age* (età) che rappresenta la sua "freschezza", ovvero l'ultima volta che questo ha manifestato una qualche attività.
L'obiettivo dell'algoritmo è che ogni nodo, scambiandosi periodicamente informazioni con gli altri mantenga la propria vista sempre aggiornata (eliminando i nodi non più attivi e aggiungendo quelli di nuova creazione) in modo da conoscere sempre la situazione corrente del sistema

2.2.1 Pseudocodice dell'algoritmo Peer Sampling

procedure ActiveThread
loop
  Wait Δ
  $j \leftarrow$ GetRandPeers(i)
  if ($PUSH$) **then**
    $buffer \leftarrow (MyAddress, 0)$
    permute($view$)
    move oldest H items to end of view
    append($buffer$,head($view$,$c/2-1$))
    send buffer to p
  else
    send ($null$) to p
  end if
  if ($PULL$) **then**
    send $buffer_p$ from p
    select($view$,c,H,S,$buffer_p$)
  end if

increaseAge($view$)
end loop
procedure PassiveThread
loop
receive $buffer_p$ from p
if ($PULL$) **then**
$buffer \leftarrow (MyAddress, 0)$
permute($view$)
move oldest H items to end of view
append($buffer$,head($view$,$c/2-1$))
send buffer to p
end if
select($view$,c,H,S,$buffer_p$)
increaseAge($view$)
end loop
method select($view$,c,H,S,$buffer_p$)
append($view$,$buffer_p$)
removeDuplicates($view$)
removeOldItems($view$,min(H,$view.size$-c))
removeHead(min(S,$view.size$-c))
removeAtRandom($view.size$-c)

2.2.2 Spiegazione e funzionamento dell'algoritmo

c	dimensione della vista parziale di ogni nodo
H	"healing" definisce quanto è aggressivo il protocollo nell'eliminare potenziali nodi non più attivi (dead links)
S	"swap" controlla il numero di descrittori scambiati tra due host

Tabella 2.2: lista dei parametri

Il protocollo è formato da due thread: uno attivo che inizia la comunicazione con gli altri nodi e uno passivo che risponde alle richieste. Sebbene il protocollo sia asincrono l'Active thread (AT) esegue le operazione ogni Δ timeslice.

Prima di tutto l'AT seleziona un vicino casualmente con il metodo *selectPeer*. Se l'informazione deve essere inviata (*push*) il buffer è inizializzato con il descrittore del nodo corrente e con i primi $c/2-1$ elementi della vista. Viene assicurato che gli elementi siano scelti casualmente e senza ripetizioni e che non ricadano tra gli H host con età maggiore. Dopodiché il buffer viene spedito. Invece, se si è in attesa di una risposta (*pull*), il buffer ricevuto viene passato come parametro al metodo *select* che, dopo aver appeso il buffer alla vista, effettua una serie di filtri per riportarla alla dimensione predefinita c:

- rimozione dei duplicati
- eventuale rimozione degli H elementi più vecchi (H indica l'aggressività del protocollo nel rimuovere gli elementi con *age* maggiore)
- eventuale rimozione degli S elementi di testa (se S è alto è molto probabile che gli elementi ricevuti siano inclusi nella vista)

Dualmente il PT effettua le stesse operazioni di ricezione e, qualora sia definito il metodo pull, le stesse operazioni di creazione del buffer e spedizione.

Considerazioni sui parametri I parametri usati nell'algoritmo di Peer Sampling, elencati nella tabella 2.3 influiscono su alcune proprietà riguardanti la propagazione e il filtraggio della vista.

- *propagazione della vista*: la strategia solamente *pull* non viene presa in considerazione in quanto, qualora tutte le connessioni in entrata di un nodo vengano terminate, questo rimarrebbe isolato. La strategia solamente *push* invece ha lo svantaggio che, qualora venga aggiunto un nuovo nodo alla rete, questo, per avere una vista aggiornata, deve attendere di essere contattato da un altro host (particolarmente svantaggioso per le reti con topologia a stella). Per questo la strategia più

efficiente (usata nei test descritti nel prossimo capitolo) è senza dubbio quella *pushpull*.

- *filtraggio della vista*: si nota immediatamente che ogni valore di H ¿ c/2 (considerando c pari) è equivalente a $H = c/2$ in quanto la vista non può avere dimensione minore di c. Per la stessa ragione $S > c/2 - H$ è equivalente a $S = c/2 - H$. Possiamo quindi identificare i tre casi descritti nella tabella 2.3.

blind	$H = 0, S = 0$	Mantiene un sottoinsieme puramente casule
healer	$H = c/2$	Mantiene la vista aggiornata in base alla freschezza
swapper	$H = 0, S = c/2$	Minimizza la perdita di informazioni

Tabella 2.3: Possibili valori dei parametri

2.3 Scelte implementative

La struttura del programma è composta da 5 thread concorrenti: due che implementano V-MAN, due dell'algoritmo di Peer Sampling e uno che gestisce l'eventuale aggiunta o rimozione di Macchine Virtuali a tempo di esecuzione.
Tutte le strutture dati dinamiche (liste) sono implementate mediante la libreria *listx.h* utilizzata nel kernel di Linux.
Ogni VM è rappresentata da un identificatore alfanumerico e dalla coppia indirizzo IPv4 e porta di provenienza: queste informazioni sono fondamentali in quanto utilizzate al termine del processo di consolidamento per effettuare la migrazione vera e propria. Ogni nodo possiede una lista (anche vuota) di VM.
Inizialmente ogni nodo legge, da un file condiviso in cui sono elencati tutti gli host presenti nella rete, il proprio *id*. Sempre dallo stesso file seleziona casualmente c host e li aggiunge alla propria vista parziale.
Ogniqualvolta avviene la comunicazione con una altro host, se il numero di

VM è cambiato, ogni nodo spedisce i descrittori delle VM (in caso di cessione) o si mette in ascolto per riceverli dall'altro (in caso di acquisto).
Per quanto riguarda la selezione dei vicini ho preferito, piuttosto che contattare tutti quelli presenti nella vista, implementare la procedure $GetRandPeer()$ che ne seleziona uno casualmente. Avendo considerato la lista come una pila (metodo First in First Out), viene privilegiata la cessione delle macchine in cima, le quali hanno probabilità maggiore di essere state precedentemente acquistate a sua volte da altri (e quindi non eseguite localmente all'host preso in esame).
Inoltre è presente un thread ausiliario che riceve gli input da tastiera permettendo di aggiungere o rimuovere, nei limiti della capacità, VM durante l'esecuzione.
Per quanto riguarda il Peer Sampling invece ogni vista parziale è stata implementata come una lista di descrittori di nodi mentre il buffer da inviare come un vettore.
Tutte le comunicazioni di rete sono state realizzate usando i Berkeley Sockets.
Infine per la gestione della concorrenza ho utilizzato una *mutex* per la vista e una per la lista di VM.

Capitolo 3

Valutazioni sperimentali

Per verificare che l'algoritmo trattato in questa tesi possieda le proprietà desiderate sono stati effettuati diversi tipi di test. Ognuno di questi serve per osservare il comportamento in determinate condizioni più o meno critiche. Soprattutto per quanto riguarda la parte che implementa V-MAN si tratta del primo test reale in quanto, come evidenziato in [15], gli unici esperimenti sono stati effettuati per mezzo di un simulatore.

3.1 L'ambiente di test

I test sono stati effettuati sulle workstation del laboratorio Ranzani del Dipartimento di Scienze dell'Informazione dell'Università di Bologna. Le macchine, oltre ad essere collegate in una rete locale, possiedono un indirizzo IP pubblico ciascuna ed eseguono *Ubuntu* 11.04. Inoltre, possedendo un account, sono tutte accessibili da remoto tramite *ssh*: per ovviare alla richiesta della password ad ogni connessione è stato sufficiente generare una coppia di chiavi pubblica/privata ed aggiungerle opportunamente ai file di configurazione.

3.2 Lo script d'avvio

Per lanciare il programma basta copiare nel file di configurazione ip_host la coppia IP_addr:port delle macchine che vogliamo far interagire inizialmente e avviare lo script:

```
./start_host.sh num_host IP port
```

dove *num_host* è il numero di righe che possono essere lette dal file ip_host per creare la vista parziale iniziale (ovviamente, essendo il file utilizzato solo in fase di bootstrap e in fase di lettura dell'identificatore non è necessario che ogni coppia IP_addr:port corrisponda effettivamente ad una workstation avviata).

Per la terminazione di uno più host è sufficiente chiudere il terminale corrispondente.

3.3 Casi di studio

In questa sezione vengono mostrati i risultati degli esperimenti a cui è stato sottoposto l'algoritmo implementato in questo elaborato per studiarne le caratteristiche fondamentali.
Su ogni host possono essere eseguite al massimo $C = 8$ VM , la dimensione della vista parziale è di $c = 4$ e $H = 1$ e $S = 2$. Inizialmente su ogni nodo sono in esecuzione un numero di VM compreso tra 0 e 8. Il valore di un Timestep (Ts) è 1 secondo.
Per ogni caso di studio sono state effettuati 5 simulazioni indipendenti.

Caso #1: interazione tra un numero costante di nodi In questo primo test osserviamo la velocità di convergenza dell'algoritmo.
Consideriamo un sistema statico dove la rete ha un numero costante di nodi $N = 10$ e le VM iniziali sono allocate casualmente.
Per ogni step $t = 1, 2, \ldots, 20$ calcoliamo $F_k(H_t)$ con $k = 0, \ldots, C$, come definita nel capitolo precedente, e rappresentiamo nel grafico nella figura 3.1

la frazione degli host con 0 VM $F_0(H_t)$ e la frazione di host completamente carichi $F_C(H_t)$.

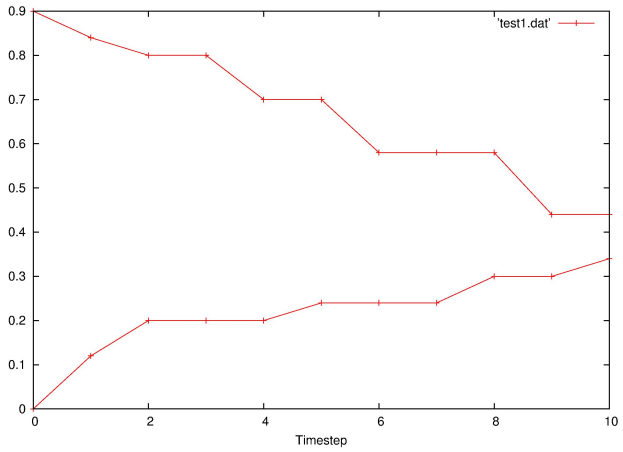

Figura 3.1: Caso 1: grafico che rappresenta l'andamento del numero delle VM eseguite durante il consolidamento: sono rappresentati rispettivamente F_0 e F_8

Dopo un tempo medio di 9,5 Ts possiamo già osservare che il consolidamento è già avvenuto e quindi potremmo in teoria già effettuare la migrazione live vera e propria: sono presenti circa $N/2$ host con nessuna VM in esecuzione che potrebbero essere spenti mentre, negli altri, sono presenti C VM. Non essendo il numero totale di VM multiplo di C rimane un host con (VM_totali mod C) VM.
Questo testimonia la rapida convergenza dell'algoritmo alla situazione ottimale.

Caso #2: failure di nodi In questo caso analizziamo il comportamento del software quando avviene lo spegnimento di più nodi nella rete.
Consideriamo un grafo con $N = 10$ nodi.
La figura 3.2 descrive la situazione iniziale della rete: sebbene i dati si rife-

riscano a una delle simulazioni effettuate, la topologia della rete e il numero delle VM è simile a quello ottenuto nelle altre 4.

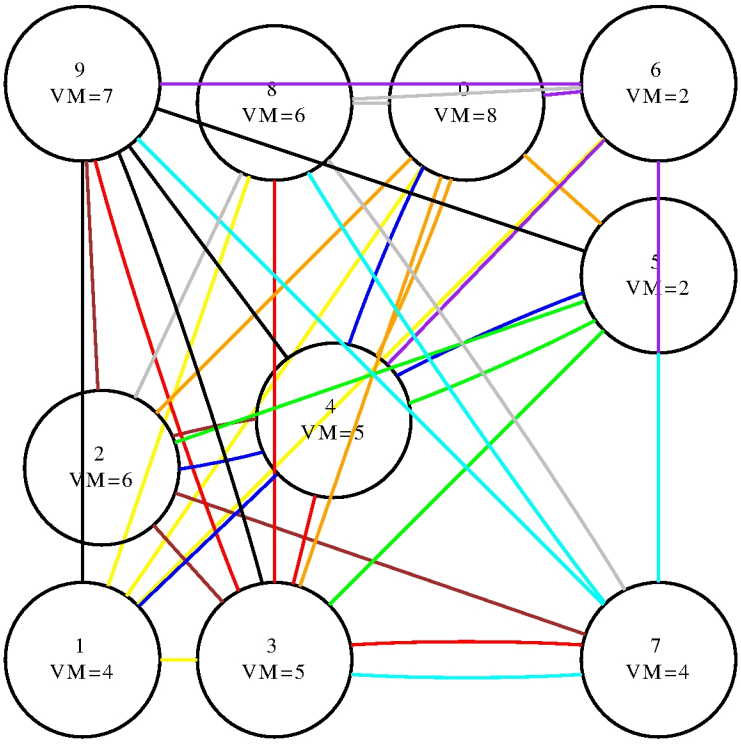

Figura 3.2: Caso 2: situazione iniziale della rete

Durante l'esecuzione, dopo che il sistema si è stabilizzato (consolidamento avvenuto), al $TS = 136$ vengono terminati i nodi con 2, 6, 8, 7.
Dopo circa 16 TS la situazione della rete è descritta della Figura 3.3.

Come si può notare la rete si è completamente stabilizzata, creando nuovi collegamenti tra i nodi rimasti.
Inoltre, sebbene le macchine in esecuzione sugli host terminati siano andate perdute, le altre sono rimaste in esecuzione sulle stesse macchine (il procedimento di consolidamento è già avvenuto precedentemente).

Caso #3: aggiunta di nodi In questo caso analizziamo il comportamento del'algoritmo quando un numero finito di nodi viene aggiunto alla rete.

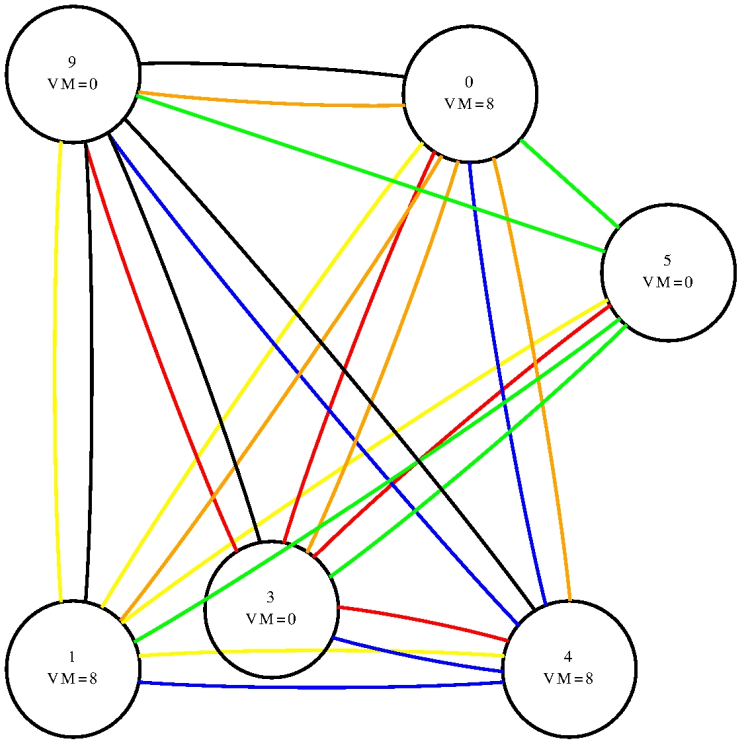

Figura 3.3: Caso 2: situazione finale della rete

Consideriamo un grafo con $N = 6$ nodi.
La figura 3.4 descrive la situazione iniziale della rete: sebbene i dati si riferiscano a una delle simulazioni effettuate, la topologia della rete e il numero delle VM è simile a quello ottenuto nelle altre 4.

Durante l'esecuzione, dopo che il sistema si è stabilizzato (consolidamento avvenuto), al $TS = 86$ vengono aggiunti i nodi 0, 1, 2, 3.
Dopo circa 15 TS la situazione della rete è descritta nella Figura 3.5.

Si nota che la rete ha assorbito a tutti gli effetti i nuovi nodi ed inoltre il processo di consolidamento, anche con l'avvio di nuove VM, in breve tempo ha creato una situazione stabile.

Caso #4: aggiunta e rimozione di VM a runtime Nell'ultimo test osserviamo il comportamento nel caso in cui vengano avviate o terminate

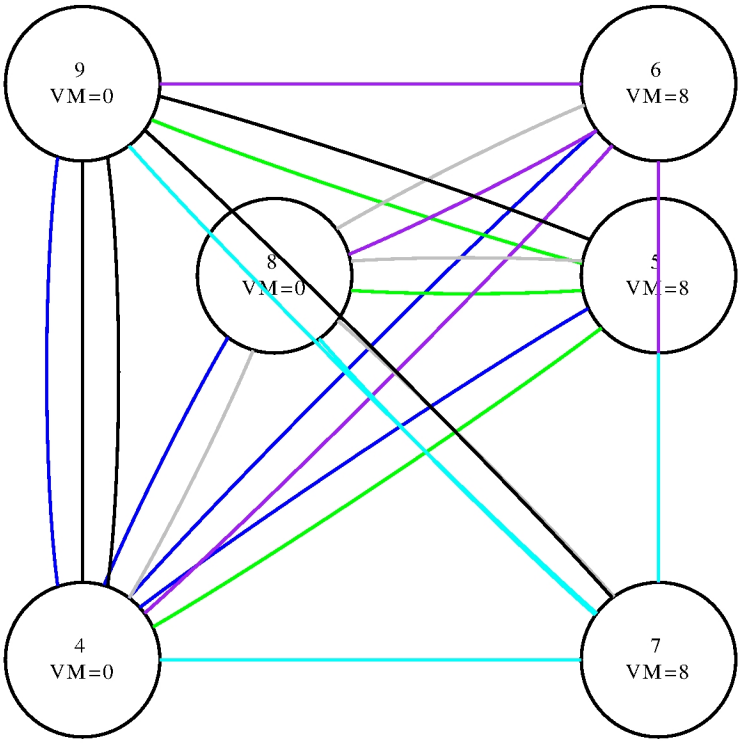

Figura 3.4: Caso 3: situazione iniziale della rete

VM durante l'esecuzione dell'algoritmo stesso.
Tutto ciò offre una simulazione abbastanza fedele della realtà in quanto, in una Cloud, gli utenti richiedono continuamente nuovi servizi per un periodo che può anche essere limitato (e quindi alla terminazione viene anche "spenta" la relativa VM).
Consideriamo un grafo con numero di nodi $N = 10$.

Inizialmente gli host vengono avviati con un numero casuale di VM.
Per ogni step $t = 1, 2, \ldots, 200$ calcoliamo $F_k(H_t)$ con $k = 0, \ldots, C$, come definita nel capitolo precedente, e rappresentiamo nel grafico nella figura 3.6 la frazione degli host con 0 VM $F_0(H_t)$ e la frazione di host completamente carichi $F_C(H_t)$.
A consolidamento avvenuto, al $TS = 72$ vengono terminate 2 VM in esecu-

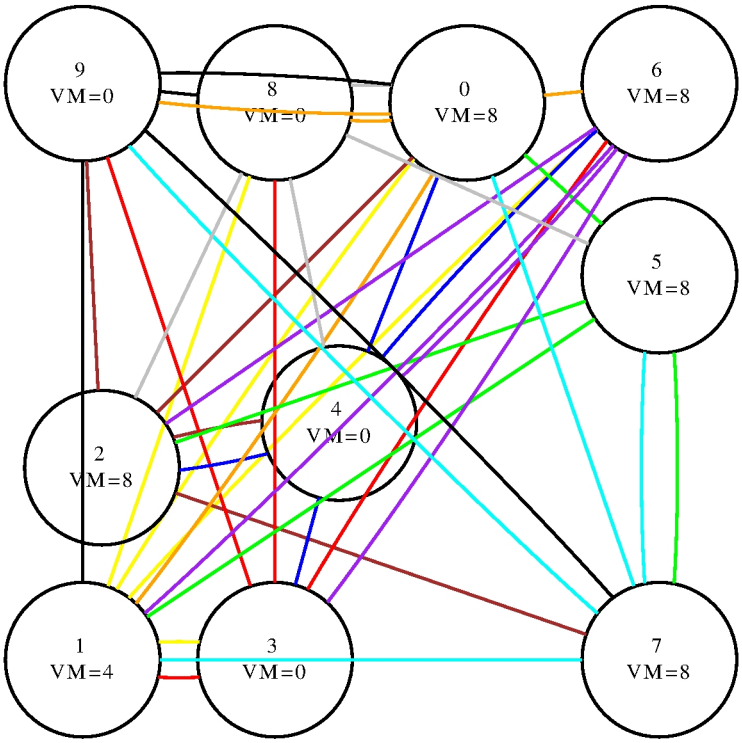

Figura 3.5: Caso 3: situazione finale della rete

zione nel nodo con $ID = 2$. Nel grafico in figura 3.6 si nota immediatamente un'improvvisa riduzione di $F_0(H_t)$ e di $F_0(H_t)$. Dopo 19 TS la situazione si è già stabilizzata.
Successivamente al $TS = 168$ vengo terminate 2 VM dell'host 3, 2 dell'host 4 e 2 dell'host 5. Anche in questo caso si nota immediatamente un'improvvisa riduzione di $F_0(H_t)$ e di $F_0(H_t)$. Dopo poco tempo (13 TS) l'algoritmo converge nuovamente verso il risultato ottimale.

3.4 Considerazioni

Gli esperimenti svolti evidenziano chiaramente la robustezza e la veloce convergenza dell'algoritmo oggetto di studio in questa tesi, anche in reti con numero di nodi e numero di VM totali variabili.

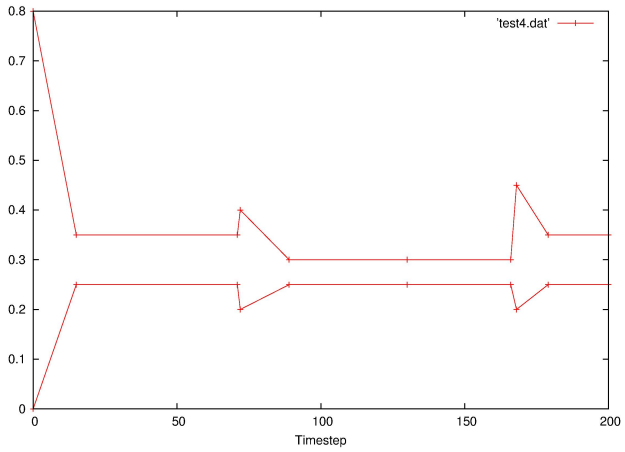

Figura 3.6: Caso 4: grafico che rappresenta l'andamento del numero delle VM eseguite durante il consolidamento: sono rappresentati rispettivamente F_0 e F_8

È bene sottolineare nuovamente che, sebbene non venga considerato il costo della migrazione live vera e propria (che a seconda della quantità di informazione da trasferire può essere più o meno esoso) questo non influisce in queste valutazioni, essendo, questa operazione, svolta solo alla terminazione volontaria del protocollo.

Conclusioni e sviluppi futuri

Lo scopo di questo elaborato è stata l'implementazione e il testing di un algoritmo per il consolidamento di macchine virtuali usabile in un ambiente di Cloud Computing.
Il vantaggio più evidente è, senza dubbio, la possibilità di massimizzare l'utilizzo delle macchine adibite a server mediante il processo di consolidamento in maniera rapida e ottimale e mantenendo attivi i servizi forniti dalla Cloud. Ottimizzando quindi l'allocazione delle Macchine virtuali si evita la presenza, in un data center, di server accesi ma sottoutilizzati .
I server in cui è in esecuzione alcuna VM possono essere spenti con la probabile diminuzione del consumo energetico.
Per la progettazione sono stati utilizzati gli algoritmi descritti in [15] e in [14].
Le valutazioni sperimentali descritte nell'ultimo capitolo evidenziano la robustezza dell'algoritmo: è resistente ad eventuali guasti dei nodi della rete e soprattutto ha un ottima risposta a aggiunta e rimozione di macchine virtuali durante l'esecuzione, scenario che rispecchia la reale variabilità di richiesta dei servizi in una Cloud.
Inoltre si nota l'elevata velocità di convergenza verso la situazione ottimale in qualunque condizione.
Sebbene la presenza di macchine virtuali sia solo simulata mediante il trasferimento delle sole informazioni identificative della macchina, questo non ne influenza la valutazione in quanto, come già specificato in più punti della dissertazione, la migrazione vera propria avviene solo al termine dell'esecu-

zione dopo un numero prestabilito di Timestep (ovviamente il costo di questa operazione può risultare molto oneroso).
Chiaramente, come possibile sviluppo futuro, può essere introdotta la presenza di macchine virtuali vere così da introdurre eventuali priorità di trasferimento in base alle loro caratteristiche (memoria utilizzate, bandwith, storage,...).
In aggiunta sarebbe possibile introdurre un meccanismo più efficiente per la fase iniziale utilizzando l'algoritmo di Bootstrapping Service [13] il quale costruisce una prima tabella di instradamento da cui partire per l'avvio di Peer Sampling.
Infine, per effettuare test su un ambiente più vicino alle realtà possibile è necessario implementarlo su uno dei software attualmente utilizzati per la gestione di una Cloud come OpenNebula, Eucalyptus e Amazon EC2.

Bibliografia

[1] Advanced configuration and power interface specification, April 5 2010. Revision 4.0a, Available at http://www.acpi.info/.

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Andrew Katz, Randy H.and Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical report, Electrical Engineering and Computer Sciences University of California at Berkeley, February, 2009.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53:50–58, April 2010.

[4] Andreas Berl, Erol Gelenbe, Marco Di Girolamo, Giovanni Giuliani, Hermann De Meer, Minh Quan Dang, and Kostas Pentikousis. Energy-efficient cloud computing. *The Computer Journal*, 53(7):1045–1051, 2010.

[5] Kenneth G. Brill. Data center energy efficiency and productivity. *High-Density Computing*, 2007.

[6] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing:vision, hype, and reality for delivering it services as computing utilities. In *2008 10th IEEE International Conferen-*

ce on High Performance Computing and Communications, pages 5–13, 2008.

[7] Changjiu, Yung-Hsiang X., Zhiyuan L., and L. Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. In *44th Annual Conf. Design Automation*, page 664–669, San Diego, CA, USA, December 2007. ACM.

[8] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proc. NSDI'05*, pages 273–286. USENIX Association, 2005.

[9] Renzo Davoli and Michael Goldweber. View-os: Change your view on virtualization.

[10] Renzo Davoli and Michael Goldweber. *Virtual Square: users, programmers and developer guide*. Lulu, 2011.

[11] Shah Devavrat. *Gossip Algorithms*, volume 3. now, 2009.

[12] Keith Jeffery and Burkhard Neidecker-Lutz. The future of cloud computing - opportunities for european cloud computing beyond 2010. Technical report, European Commission, 2010.

[13] Màrk Jelasity, Alberto Montresor, and Ozalp Babaoglu. The bootstrapping service. In *26th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW'06)*, page 11, 2006.

[14] Màrk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Kermarrec Anne-Marie, and Maarten Van Steen. Gossip-based peer sampling. *ACM Trans.*, 25, August August 2007.

[15] Moreno Marzolla, Ozalp Babaoglu, and Fabio Panzieri. Server consolidation in clouds through gossiping. Technical report, Department of Computer Science University of Bologna, January 2011.

[16] Mell Peter and Timothy Grance. The nist definition of cloud computing (draft). *NIST Special Publication*, January 2011.

[17] James E. Smith and Ravi Nair. *Virtual Machines: versatile platforms for systems and processes*. Morgan Kaufmann, 2005.

Ringraziamenti

In primo luogo desidero ringraziare il dottor Moreno Marzolla per avermi offerto supervisione e consigli nello svolgimento di questo lavoro e soprattutto per aver sopportato e corretto i miei vari errori.
Desidero inoltre ringraziare i miei genitori, Roberto e Antonella e mia sorella Silvia per il supporto morale durante questa prima parte di carriera accademica.
Ringrazio tutti i miei colleghi con cui ho passato importanti momenti di studio e di svago rendendo piacevole e avventuroso il percorso durante questi tre anni.
Infine ringrazio tutti i miei coinquilini passati e presenti che hanno reso entusiasmante la vita da studente fuorisede.