Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Informatica

Progettazione ed implementazione
di una
piattaforma di cloud computing
per il
supporto ad applicazioni VoIP

Tesi di Laurea in
Architettura degli Elaboratori

Relatore:
Chiar.mo Prof.
Ghini Vittorio

Correlatore:
Dott.
Ferretti Stefano

Presentata da:
Piccinelli Flavio

Sessione III
Anno Accademico 2010-2011

Indice

Introduzione

Nell'ambito della Information Technology, si va sempre più affermando l'utilizzo di infrastrutture di "cloud computing" (letteralmente nuvola di computazione); ovvero un insieme di tecnologie che permettono ad un utente di accedere a risorse (unità di calcolo piuttosto che di archiviazione) distribuite sulla rete, o meglio, localizzate nella "nuvola" appartenente al provider che offre il servizio.

La rapida crescita di queste tecnologie è sicuramente favorita da due fattori: da un lato abbiamo la drastica diminuzione dei costi di gestione e manutenzione che comportano le soluzioni basate sul cloud computing; dall'altro lato abbiamo l'incredibile diffusione, soprattutto negli ultimi anni, di dispositivi mobili sufficientemente elaborati per accedere a servizi internet, ma con limitate capacità di calcolo e di archiviazione, ovvero, i perfetti clienti di applicazioni basate su questo modello.
L'utilizzo attuale dei sistemi di cloud computing, però, si limita, come detto prima, a consentire ai terminali mobili di accedere alle risorse distribuite, non permettendo, per esempio, l'utilizzo contemporaneo delle molteplici interfacce di rete di cui sono dotati i moderni dispositivi mobili.

Questo progetto di tesi, basato su quello di Vincenzo Tilotta [21], si pone quindi come obiettivo, progettare una piattaforma basata su una soluzione di cloud computing per supportare applicazioni VoIP[1], implementando

[1]VoIP acronimo di Voice over IP; tecnologia che rende possibile effettuare una conversazione telefonica sfruttando una connessione Internet.

il modello ABPS (Always Best Packet Switching) per sfruttare al meglio le potenzialità dei moderni terminali come *smartphone* o *tablet*.

In questo elaborato inizieremo con una panoramica e una descrizione generale di cloud computing, dei benefici e degli svantaggi che presenta e dei protocolli utilizzati. Passeremo poi ad esporre la piattaforma da noi creata, lo scenario d'uso, la sua progettazione ed implementazione. Infine vedremo com'è stata testata ed i possibili sviluppi futuri.

Elenco delle figure

Elenco dei sorgenti

Capitolo 1

Il cloud computing

Il cloud computing, come definito dal NIST[1][11], è un modello che permette l'accesso on-demand e ovunque ci si possa connettere a internet, ad un insieme di risorse, come reti, server, applicazioni e servizi, richiedendo agli utenti il minimo sforzo di gestione dell'infrastruttura; esso permette quindi di "estendere" le funzionalità di un qualsiasi dispositivo che possa connettersi alla rete.

Questo modello a *nuvola* si compone di cinque caratteristiche essenziali (che vediamo subito), tre modelli di servizio e quattro modelli di distribuzione.
Le caratteristiche basilari sono:

Self-service on-demand Gli utenti possono richiedere risorse, come capacità di calcolo o archiviazione, quando ne hanno bisogno ed essere soddisfatti senza richiedere l'interazione manuale con ogni provider di servizio.

Accesso tramite la rete Le risorse sono accessibili da qualunque punto della rete internet attraverso protocolli e meccanismi standard, così da permetterne l'utilizzo da parte di un insieme eterogeneo di dispositivi (come ad esempio computer, smartphone, tablet, etc).

[1]National Institute of Standards and Technology, agenzia del governo degli Stati Uniti d'America che si occupa della gestione delle tecnologie

Raggruppamento delle risorse Le capacità computazionali del provider sono agglomerate per servire molteplici utenti contemporaneamente utilizzando il modello multi-tenant (letteralmente multi-cliente), con svariate risorse fisiche e virtuali assegnate e riassegnate dinamicamente in base alle richieste degli utenti. Generalmente, in questo modo i clienti non sanno e non hanno il controllo sulla locazione fisica delle risorse o del servizio richiesto, è comunque possibile che il provider permetta agli utenti di specificare il luogo in modo astratto, come città, stato o datacenter.

Elasticità Le risorse possono essere allocate e rilasciate dinamicamente, in alcuni casi automaticamente, per permettere di commisurare rapidamente i mezzi impegnati in base alla domanda. Dal punto di vista del cliente le risorse disponibili spesso appaiono illimitate ed esso può richiederne in qualsiasi momento e quantità.

Servizio misurato I sistemi cloud controllano e ottimizzano automaticamente l'uso delle risorse implementando un sistema di monitoraggio e accounting ad un livello di astrazione appropriato al tipo di servizio (utilizzo di spazio di archiviazione, di CPU o di banda). L'utilizzo delle risorse può essere monitorato, controllato e rendicontato, offrendo trasparenza e chiarezza sia nei confronti del provider, sia degli utenti che usufruiscono del servizio.

Nelle prossime sezioni vedremo le altre due componenti.

1.1 Tre modelli di nuvola

Come abbiamo detto, sono stati definiti [11, 13] tre modelli di servizio, ovvero tre tipologie di infrastruttura (e conseguentemente diversi servizi) che il provider può fornire agli utenti (per uno schema si veda in fig. 1.1):

Infrastructure as a Service o IaaS I mezzi messi a disposizione degli utenti sono: capacità di calcolo, di archiviazione, network e altre risorse

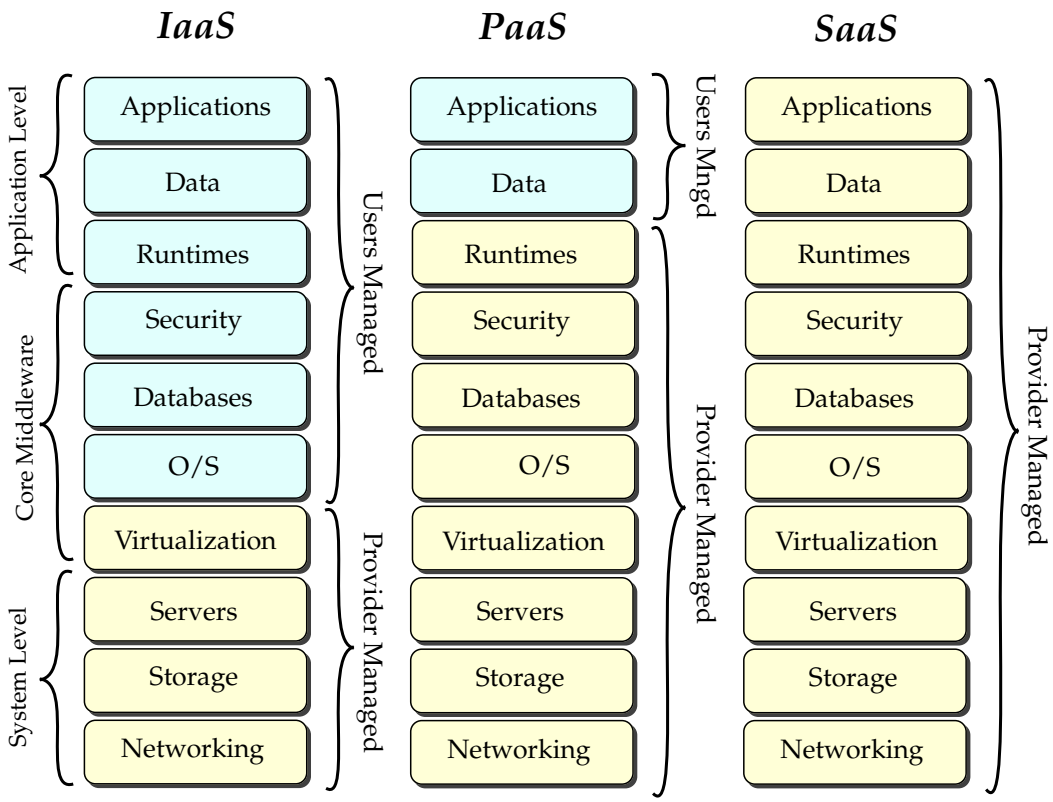

Figura 1.1: Schema delle responsabilità nel cloud computing

hardware basilari sulle quali i clienti sono abilitati ad eseguire software arbitrariamente scelto, che può includere applicazioni utente o interi sistemi operativi. Il cliente non gestisce direttamente l'infrastruttura hardware sottostante, ma ha il completo controllo su tutto ciò che ha deciso di eseguire (sistema operativo, applicazioni e spazio di archiviazione) e, in alcuni casi, può avere un limitato controllo su alcuni componenti della rete, come ad esempio il firewall.

Platform as a Service o PaaS I mezzi messi a disposizione dei clienti sono fondamentalmente: l'hardware, il sistema operativo e le librerie. Su questi strumenti gli utenti possono eseguire programmi da loro creati o acquistati, e che utilizzano librerie, servizi o tool supportati (o com-

patibili con quelli supportati) dal provider. Il cliente non può gestire l'infrastruttura sottostante, come le reti, i server, il sistema operativo, etc., ma ha il controllo solamente sulle impostazioni delle applicazioni che ha lanciato.

Software as a Service o SaaS In questo modello gli utenti posso solamente utilizzare le applicazioni fornite dal provider ed eseguite all'interno dell'infrastruttura. Le applicazioni, solitamente, sono accessibili dai clienti tramite interfacce web o applicazioni client ad-hoc. L'utente non può in alcun modo gestire o modificare l'infrastruttura sottostante, con l'unica eccezione di poter settare alcune preferenze relative all'applicazione che sta utilizzando.

1.1.1 Alcuni esempi

Per meglio comprendere le differenze tra questi tre modelli vediamo ora alcuni esempi, o meglio uno per modello:

IaaS Amazon EC2 [1] è un esempio di modello Infrastructure as a Service. Tramite questa piattaforma, infatti, Amazon permette agli utenti registrati di "affittare" i suoi server, su cui possono lanciare programmi o interi sistemi operativi. Come è descritto sul sito, i clienti pagano in base al tempo di utilizzo, alla quantità di CPU utilizzata e alla quantità di memoria richiesta.

PaaS Google App Engine [6] è una nuova funzionalità di Google che permette, appunto, di scrivere e lanciare applicazioni su server remoti; per ora supporta i linguaggi Java e Python.

SaaS A questa categoria appartengono molte applicazioni web-based, come ad esempio le Google Apps e tutte quelle applicazioni a cui è possibile accedere tramite un browser web.

1.2 I modelli di distribuzione

Come abbiamo detto all'inizio, esistono quattro modelli di distribuzione, ovvero modelli per disegnare ed implementare la propria infrastruttura; andiamo a definirli [11]:

Private cloud L'infrastruttura è pensata per essere utilizzata da una singola organizzazione che comprende molteplici utenti (per esempio più unità commerciali). Essa può essere posseduta, mantenuta e gestita dall'organizzazione in questione, da una terza parte o una combinazione delle due.

Community cloud La piattaforma è strutturata per essere utilizzata da una specifica comunità di utenti, appartenenti a diverse organizzazioni che hanno deciso di operare insieme. Essa può essere posseduta, mantenuta e gestita da una o più organizzazioni appartenenti alla comunità, da una terza parte o una combinazione di queste.

Public cloud L'infrastruttura è indirizzata ad un pubblico generico. Essa può essere posseduta, mantenuta e gestita da un'azienda, da un'organizzazione accademica o governativa, da una terza parte o una combinazione di queste.

Hybrid cloud Questo tipo di infrastruttura, come dice il nome stesso, è una commistione di più piattaforme di cloud computing distinte (private/comunitarie/pubbliche) che rimangono entità distinte, ma sono legate tra loro da una tecnologia condivisa - standard o meno - che rende possibile la condivisione e la portabilità di dati ed applicazioni.

1.3 I pregi e i difetti del cloud computing

Le soluzioni basate sul cloud computing presentano molti vantaggi, ma, come vedremo di seguito, anche svantaggi e, curiosamente, alcuni punti di forza possono coincidere con i punti deboli.

Innanzitutto, questo modello di business, potrebbe cambiare radicalmente la struttura delle imprese, sia legate all'IT, che non [15]; dal punto di vista di un'azienda che vuole iniziare una nuova attività, appoggiarsi ad un'organizzazione che offre infrastrutture cloud, riduce notevolmente sia i costi fissi iniziali (non deve acquistare l'hardware), sia i costi di manutenzione dell'infrastruttura (software e hardware), essendo questi a carico del provider di servizi cloud.
Anche gli utenti, come abbiamo già detto, hanno la possibilità di usufruire di servizi complessi tramite dispositivi relativamente semplici, come smartphone o tablet, ed avere la comodità di poter accedere ai propri dati da qualsiasi punto sulla rete (un esempio è iCloud di Apple).

Per quanto riguarda la sicurezza, da un lato possiamo pensarlo come un punto di forza: una grande azienda come Google, molto probabilmente, ha dei sistemi di sicurezza molto più robusti di un normale computer desktop. Dall'altro lato, alcuni [8] lo vedono come un punto di debolezza: i dati salvati nella nuvola potrebbero essere tutt'altro che sicuri, ovvero bisognerebbe fidarsi del provider, il quale, volendo, potrebbe utilizzare queste informazioni o altri dati personali in modo illecito senza che l'utente possa intervenire. Inoltre, tutti i dati scambiati con il provider attraversano la rete pubblica e, per questo, possono essere soggetti ad azioni di *sniffing*[2] o *hacking*.

Altro nodo ambiguo è il concetto di continuità del servizio, dato che, da una parte, gli utenti sono fortemente dipendenti dal provider, sia per quanto riguarda l'integrità dei dati, sia perché c'è la possibilità che il servizio possa essere temporaneamente non disponibile per problemi imputabili al provider. Dall'altra parte, però, un provider *responsabile* fa tutto ciò che è in suo potere per garantire l'integrità dei dati e la continuità del servizio, come ad esempio conservare copie ridondanti dei dati o sovrastimando il carico

[2]Termine con cui ci si riferisce all'attività di intercettare la comunicazione al fine di sottrarre informazioni

di lavoro, rendendo in questo modo la nuvola più sicura del computer di casa.

Capitolo 2

I protocolli orientati al cloud computing

Vediamo ora alcuni protocolli, standard e non, utilizzati nell'ambito delle tecnologie web ed in particolare quelli relativi alle soluzioni basate sul cloud computing.

2.1 Il VoIP

Dato che lo scopo principale della nostra piattaforma di cloud computing è supportare servizi VoIP, vogliamo ora definire meglio cos'è e come funziona.

Con il termine VoIP, acronimo di Voice over Internet Protocol, si intende una tecnologia che permette di effettuare una telefonata sfruttando una qualsiasi connessione TCP/IP, ovvero la rete internet. Il VoIP è un insieme di protocolli a livello applicativo, in particolare i protocolli di comunicazione RTP, SIP e H.323[1].

Questa tecnologia, se confrontata con la telefonia tradizionale, presenta molti vantaggi: ad esempio, sfruttando la commutazione di pacchetto,

[1]Ha la stessa funzione del protocollo SIP, da cui è ormai stato superato

non obbliga a riservare una porzione di banda prefissata per ogni chiamata, sfruttando al meglio le risorse della rete. Abbiamo inoltre un minor costo di manutenzione delle infrastrutture (è sufficiente la rete IP sia per le comunicazioni voce che per quelle dati) ed una maggiore modularità, poiché se si vogliono implementare funzionalità nuove, non sarà necessario cambiare l'hardware ma solamente il software.

2.1.1 I protocolli RTP e SIP

RTP (Real-time Transport Protocol) e SIP (Session Initiation Protocol) sono i due protocolli di comunicazione alla base di una chiamata VoIP (ovviamente sopra il protocollo TCP/IP): il primo, in fase di standardizzazione, si occupa del trasporto di pacchetti voce su IP, il secondo, standard IETF[2], è preposto alla gestione a livello più alto della comunicazione, principalmente della segnalazione di una chiamata al destinatario e la ricostruzione e sincronizzazione dei frame audio.

2.2 VNC e il protocollo RFB

I programmi di Virtual Network Computing (VNC) sono strumenti che permettono l'accesso ed il controllo di una macchina remota tramite condivisione del desktop, sfruttando il protocollo standard RFB (Remote Framebuffer) [16]. Tramite questo protocollo, quindi, è possibile accedere a distanza a interfacce grafiche, come appunto il desktop, sia esso di una macchina fisica o virtuale.
Il programma VNC si occupa quindi di trasmettere al computer gli eventi del mouse e della tastiera ed attende che l'RFB aggiorni il desktop modificato.

Come è facile intuire, questa tecnologia è necessaria per utilizzare una macchina virtuale (non avendo questa un desktop fisico) lanciata su un

[2]Acronimo di Internet Engineering Task Force, è una comunità aperta che si occupa di sviluppare e promuovere standard internazionali nell'abito delle tecnologie connesse ad Internet

server remoto; questa soluzione è sfruttata anche nell'infrastruttura da noi progettata.

Il difetto più rilevante del VNC deriva dal fatto che il protocollo RFB, basato sul TCP, non offre nessuno strumento di crittazione della connessione; il desktop remoto può essere protetto solamente da una password di otto caratteri che, all'atto pratico, potrebbe essere facilmente individuata: basterebbe intercettare la chiave di criptazione e la password codificata.
Per colmare questa lacuna, è stato definito il protocollo VeNCrypt, un protocollo non standard che incapsula la connessione in un tunnel TLS; oppure è possibile collegarsi al server VNC tramite un tunnel SSH. Di seguito parleremo di questi protocolli orientati alla sicurezza.

2.3 I protocolli per la sicurezza

Ora vedremo alcuni dei protocolli preposti a garantire la riservatezza di una comunicazione, in particolare andiamo a descrivere brevemente i tre protocolli più importanti su cui si basa la sicurezza della nostra infrastruttura.

2.3.1 TLS

TLS, o Transport Layer Security [4], è un protocollo standard IETF che permette di stabilire una connessione sicura (autenticata e riservata) tra due nodi su una rete TCP/IP (esso è situato in un livello intermedio tra quelli applicativo e di trasporto).
Il protocollo è basato su due tecniche di crittografia: quella asimmetrica (tipicamente basata su chiave privata RSA e certificato x509) e quella simmetrica (per una descrizione più dettagliata si veda [3, Caressa 2005]). Al fine di iniziare una connessione riservata, il client contatta il server, iniziando il cosiddetto *handshake TLS*, composto dalle seguenti fasi:

1. Negoziazione dell'algoritmo di cifratura simmetrica e di hash da utilizzare.

2. Invio del certificato al client contenente identità e chiave pubblica del server.

3. Generazione della chiave condivisa (spesso utilizzando l'algoritmo AES o triple DES).

4. Cifratura simmetrica dei pacchetti, garantendo la riservatezza della comunicazione.

2.3.2 SSH

L'SSH, acronimo di Secure Shell [12], è un protocollo di livello applicazione che permette di stabilire una connessione sicura ed autenticata ad una macchina remota tramite interfaccia a riga di comando. Questo protocollo è, di fatto, l'evoluzione di Telnet, il quale ha funzionalità analoghe, ma la connessione che stabilisce non è cifrata.

OpenNebula, come vedremo nel capitolo 3, utilizza il protocollo SSH per gestire l'intera infrastruttura cloud, lanciando comandi da riga di comando tramite il protocollo suddetto. Eseguire comandi su un computer remoto tramite interfaccia SSH è molto semplice; è sufficiente impartire la seguente direttiva sulla macchina locale:

```
$ ssh [opzioni] utente@macchina.remota [comando]
```

il server remoto richiederà al client di autenticarsi (richiedendo la password utente o una chiave RSA valida) e lancerà sulla "macchina.remota" il comando specificato.

2.3.3 VeNCrypt

Come abbiamo visto nella sezione precedente, VeNCrypt [22] è un protocollo open source non standard, utilizzato dai programmi VNC per rendere sicuro il protocollo RFB.
Questo protocollo, non essendo standard, è supportato solo da alcuni VNC server e client; in particolare è supportato dal VNC server implementato da

Qemu (come vedremo in sezione 3.1.1) e dai VNC client basati su gtk-vnc [20].

VeNCrypt prevede sette livelli di sicurezza, che vanno dalla sicurezza *plain*, ovvero utilizzando l'autenticazione tramite coppia username e password, fino ad un livello di sicurezza piuttosto alto chiamato *x509Plain*, ovvero in cui il server ed il client si autenticano a vicenda tramite certificati x509, creano un tunnel TLS, ed infine il client accede al desktop desiderato immettendo nome utente e password[3].

[3]Nella nostra piattaforma questo non è possibile: se si impone l'inserimento della password il sistema non crea il tunnel TLS; questo probabilmente è dato dal fatto che il server VNC di Qemu e il client VNC da noi usato implementano versioni diverse di VeNCrypt

Capitolo 3

OpenNebula

OpenNebula è un toolkit, o set di strumenti software, open source per la gestione di centri di elaborazione dati distribuiti eterogenei; ovvero un toolkit per la gestione di un'infrastruttura di cloud computing.

Due sono i motivi che ci hanno portato alla decisione di basare la nostra piattaforma su questa infrastruttura.

Innanzitutto, come viene detto nel rapporto della Commissione Europea sul futuro del cloud computing [18], nell'ambito dei progetti di cloud computing OpenNebula (assieme ad OpenStack) è probabilmente il più prominente. In secondo luogo, OpenNebula fornisce molteplici funzioni per l'integrazione, la gestione, e la sicurezza delle risorse. Inoltre si focalizza sulla standardizzazione e sulla interoperabilità; questo toolkit, infatti, fornisce agli amministratori dell'infrastruttura diverse interfacce (EC2 Query, OGF OCCI e vCloud) e hypervisor (Xen, KVM e VMware) oltre ad una architettura molto flessibile in grado di creare Cloud private, pubbliche ed ibride, permettendo di utilizzare configurazioni hardware molto complesse.

La struttura ed il funzionamento di OpenNebula sono apparentemente molto semplici; come mostrato in figura 3.1 è presente un server o "front-end" - tramite il quale è possibile gestire i nodi appartenenti all'infrastruttura

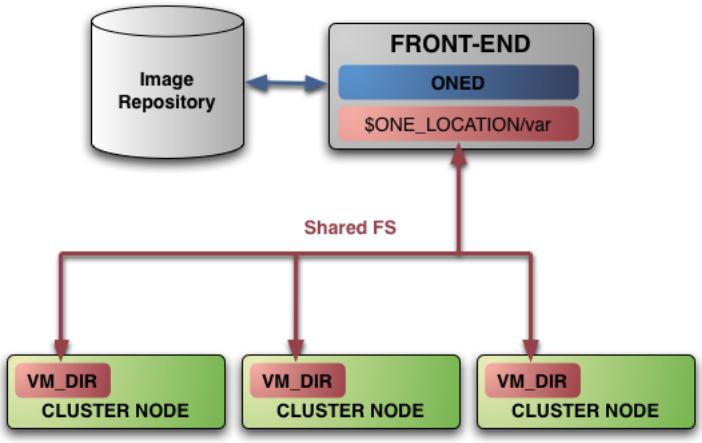

Figura 3.1: Struttura con NFS

e le fasi della virtualizzazione - e vari nodi di calcolo, ovvero i server fisici sui quali verranno lanciate le istanze delle macchine virtuali.

Le funzioni implementate permettono di aggiungere o togliere nodi dalla nostra infrastruttura, e lanciare, cancellare o migrare le macchine virtuali sui vari nodi. Tutte queste operazioni vengono effettuate lanciando comandi ssh dal front-end sui vari nodi.

3.1 Lo stack di OpenNebula

Le varie componenti di OpenNebula possono essere rappresentate da una pila come quella in figura 3.2. Come si può vedere, OpenNebula utilizza le interfacce fornitegli dai componenti inferiori, come i driver di virtualizzazione (nel nostro caso qemu-kvm) o di trasferimento delle macchine virtuali (nel nostro caso nfs) e fornisce agli amministratori dell'infrastruttura le API per gestire la stessa.

La nostra piattaforma utilizza essenzialmente le API per ruby, i driver nfs[1]

[1]Acronimo di network file system, ovvero un file system condiviso

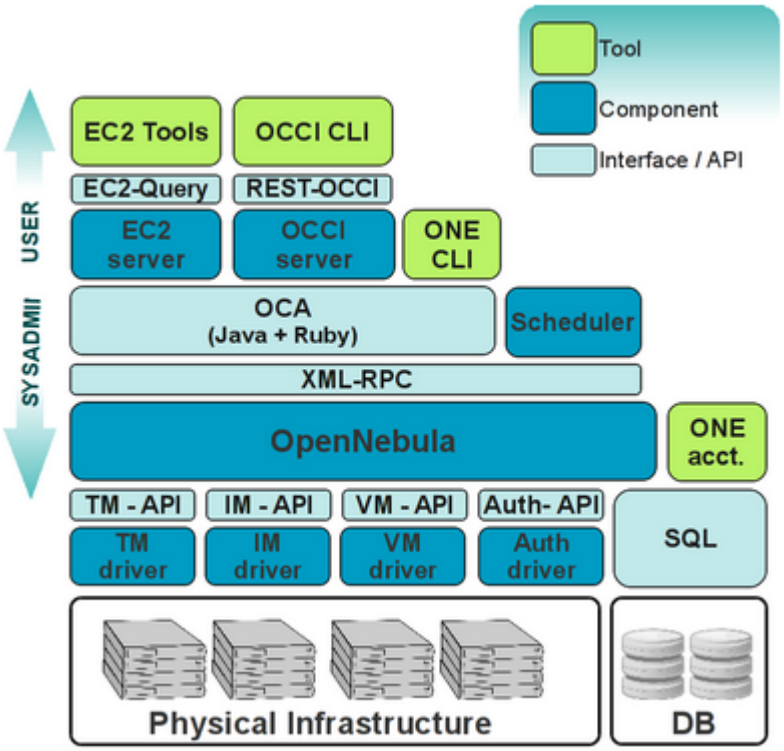

Figura 3.2: Stack di OpenNebula

(avendo tutti nodi della nuvola il repository delle macchine virtuali condiviso) e, come detto prima, i driver qemu-kvm.

Per rendere più chiaro il funzionamento di OpenNebula descriveremo in dettaglio i driver qemu-kvm.

3.1.1 Qemu-Kvm

KVM, acronimo di Kernel-based Virtual Machine [9], è un'infrastruttura per la completa virtualizzazione del kernel Linux, utilizzando le estensioni di virtualizzazione hardware come Intel VT o AMD-V. Esso consiste in un modulo kernel che fornisce l'infrastruttura di base e in un modulo, specifico per il processore della macchina fisica (Intel o AMD), per la virtualizzazione della CPU.

Per poter utilizzare il modulo KVM è necessario Qemu, un software di emulazione hardware che, grazie alla tecnica della traduzione dinamica, è molto performante [2].
Qemu è in grado di aumentare notevolmente le sue prestazioni di emulazione tramite la virtualizzazione kernel, utilizzando il suddetto modulo KVM. Quest'ultimo è il nostro caso, dato che lanciamo macchine ospiti, con sistema operativo Linux x86, su macchine host con il medesimo sistema.

Qemu implementa anche un server VNC, così da permettere l'accesso alle macchine virtuali anche da remoto, senza dover cedere al client il controllo del desktop fisico della macchina host; questa è l'unica soluzione che permette ad utenti distinti di accedere contemporaneamente a più macchine virtuali lanciate su uno stesso server.

Come vedremo nella prossima sezione, l'infrastruttura Qemu-Kvm è richiamata ed utilizzata da OpenNebula tramite il demone libvirtd.

3.1.2 Libvirt

Libvirt è un'API e demone (libvirtd) [10] utilizzato per gestire varie piattaforme di virtualizzazione come Xen, Qemu-Kvm e molti altri. OpenNebula, per gestire le macchine virtuali, utilizza principalmente le API di Libvirt; uno schema riassuntivo è mostrato in figura 3.3.

La sequenza delle operazioni eseguite da OpenNebula e da Libvirt è la seguente:

1. OpenNebula
   (a) Parsing del file di definizione della macchina virtuale (template)
   (b) Creazione del file di configurazione in formato XML di Libvirt
2. Libvirt
   (a) Parsing del file di configurazione

(b) Creazione della stringa per lanciare Qemu-Kvm

(c) Lancio di Qemu-Kvm tramite il demone libvirtd

Per maggiori dettagli si veda la documentazione di OpenNebula [14].

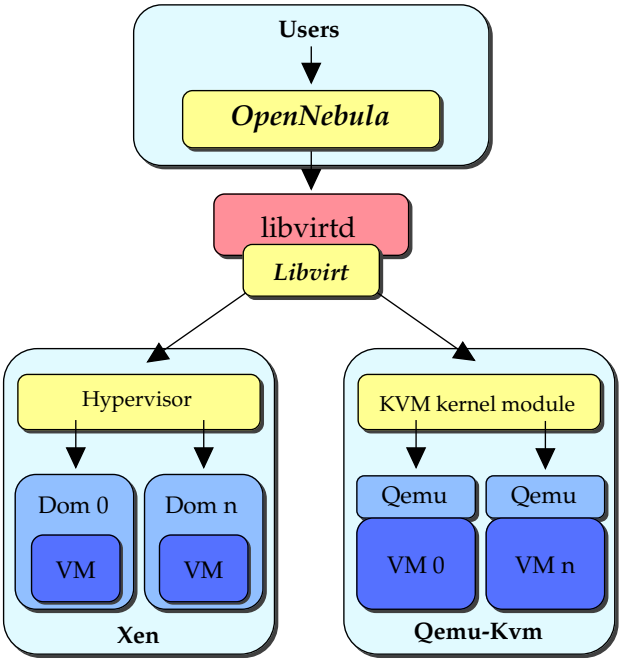

Figura 3.3: Schema delle chiamate per la virtualizzazione

Capitolo 4

Lo scenario

Vediamo ora l'ambito di applicazione del nostro progetto. Come detto nell'introduzione, oggigiorno vi è una grande diffusione di terminali mobili sofisticati ma, a causa dell'esigua disponibilità di energia, non molto performanti: ridotta capacità di calcolo e limitata memoria di massa. Per questo motivo, per utilizzare funzionalità complesse e avide di risorse, è necessario ricorrere ad un approccio chiamato Mobile Cloud Computing (MCC)[5], che permette agli utenti di dispositivi mobili di superare queste limitazioni; demandando il carico di lavoro maggiore a server condivisi sulla rete, messi a disposizione da provider di servizi di cloud computing.

4.1 Il cloud computing per il VoIP

La nostra piattaforma, nello specifico, si prefigge come obiettivo l'erogazione dinamica di proxy server virtuali per la fornitura di servizi a dispositivi mobili, in particolare, servizi VoIP. Per uno schema esemplificativo del dominio si veda figura 4.1.

La necessità di avere proxy server *dinamici* deriva principalmente da due fattori. Innanzitutto permette di avere un'infrastruttura sufficientemente

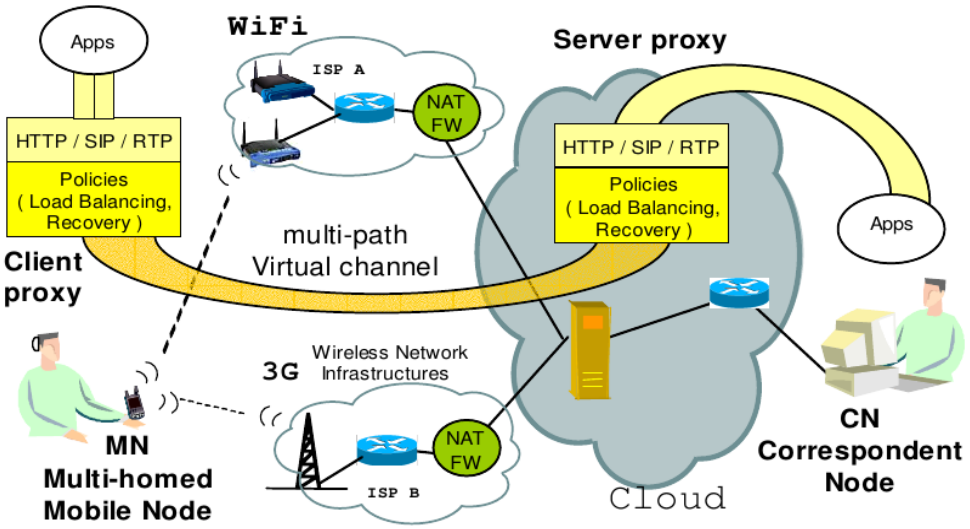

Figura 4.1: Mobility management architecture

scalabile, al fine di gestire un grande numero di utenti; le risorse dei proxy server, infatti, sono allocate on-demand, ovvero solo quando un terminale richiede il servizio.
In secondo luogo, permette di sfruttare al meglio le molteplici interfacce di rete wireless dei dispositivi mobili moderni.

Questa seconda funzionalità è molto importante, se non necessaria, per il corretto funzionamento dei servizi che richiedono una connessione continua tra il terminale e il provider del servizio (come ad esempio un programma VoIP), quando l'utente è in movimento.
In questo caso, il dispositivo mobile deve continuamente controllare la qualità della connessione in corso e, in caso sia scarsa, deve cercare di stabilire una nuova connessione tramite una delle altre interfacce di rete.
Per fare un esempio, pensiamo ad uno smartphone, dotato di antenna *3G* e *wi-fi*: l'utente inizia la connessione col server quando è connesso ad un access point wi-fi, se si allontana troppo, il telefono deve cercare di stabilire una nuova connessione col provider tramite la rete 3G o vice versa.

Per eseguire queste operazioni (controllo della rete - ricerca di un nuovo access point - stabilimento di una nuova connessione) esiste il protocollo MIH,

acronimo di Media-Independent Handover. Esso è uno standard IEEE 802.21 in fase di sviluppo avanzata [7].

Capitolo 5

Progettazione della piattaforma

La nostra piattaforma è stata progettata pensando ad un provider di servizi di cloud computing, ed in particolare di erogazione di macchine virtuali a supporto di servizi VoIP.
Essa consta di quattro elementi:

- Il web server[1], utilizzato per registrare nel sistema i nuovi utenti, e inserire nel database lo username e la password tramite cui si autenticheranno.

- Il dispatcher, ovvero il *monitor* dell'infrastruttura; è il server tramite cui è possibile monitorare e gestire l'intera piattaforma, per bilanciare il carico di lavoro tra i vari nodi di calcolo dell'infrastruttura, e a cui gli utenti si connettono per richiedere l'erogazione del servizio.

- I nodi del cluster, ovvero i centri di calcolo veri e propri che eseguiranno le istanze delle macchine virtuali.

- L'applicazione client che l'utente esegue per connettersi al dispatcher e richiedere il servizio desiderato.

[1]Attualmente non implemetato, ma simulato tramite un semplice script Ruby.

Nella prossima sezione andremo a definire dettagliatamente i ruoli di questi quattro blocchi funzionali e come interagiscono tra loro seguendo il protocollo da noi ideato.

5.1 Il protocollo

Innanzitutto, prima di poter usufruire del servizio, gli utenti devono registrarsi al sistema; operazione che deve essere effettuata connettendosi al web server (tramite protocollo https). Essi devono poi scegliere username e password tramite cui si autenticheranno al dispatcher. Una volta fatto questo, possono connettersi al server dell'infrastruttura e richiedere il servizio desiderato.

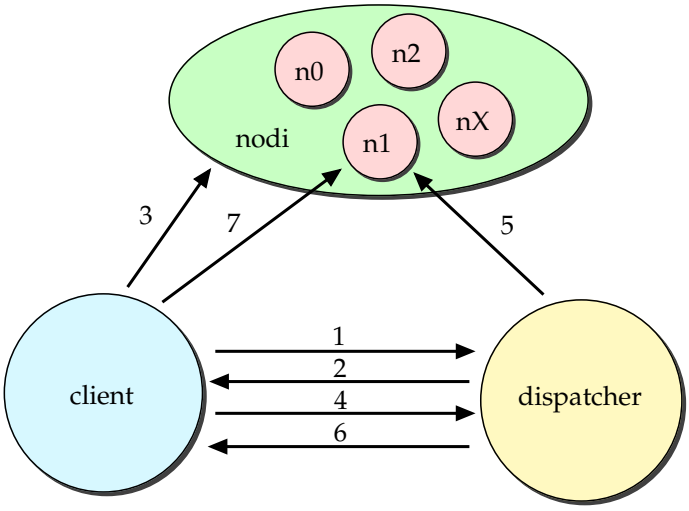

Figura 5.1: Schema del protocollo

Vediamo ora, con l'ausilio dello schema in figura 5.1, la sequenza, in ordine temporale, delle operazioni compiute da client e dispatcher durante l'esecuzione del protocollo:

1. Il client si connette al server, si autentica con username e password, iniziano una sessione TLS e l'utente invia al server la richiesta di erogazione del servizio.

2. Il dispatcher controlla quali nodi sono disponibili e invia al client la lista dei cinque nodi meno carichi.

3. Il client identifica il calcolatore più vicino tra i cinque e ...

4. ... invia al dispatcher il nome del nodo desiderato.

5. Il server lancia il servizio desiderato sul nodo selezionato (in questo caso il nodo "n1") e ...

6. ... comunica al client le porte tcp/udp per connettersi al server.

7. Infine il client si connette al servizio richiesto.

In questo elenco abbiamo parlato di nodi *meno carichi* e nodo *più vicino*; vediamo ora cosa significa in pratica. Il carico di un nodo è calcolato in base ai bogomips disponibili [23], ovvero un valore assoluto che indica le potenzialità del processore non allocate ad altri incarichi.
La distanza tra l'utente ed i nodi è calcolata, dal client stesso, facendo un ping verso i computer indicatigli dal dispatcher, ovvero inviando pacchetti *ICMP* e calcolando la latenza, ovvero quanti millisecondi trascorrono tra la richiesta e la risposta, il nodo che risponde più velocemente è il più vicino.

5.2 La sicurezza

Questo concetto è cruciale, soprattutto nell'ambito dei servizi remoti, in quanto, come abbiamo già visto, tutte le comunicazioni, e quindi tutti i dati scambiati tra l'utente ed il provider, sono inviati attraverso la rete pubblica; inoltre, dovendo il sistema gestire una molteplicità di utenti, è necessario garantire l'accesso solamente, e a tutti, i clienti registrati e ogni utente deve poter accedere solamente al servizio da lui richiesto senza poter interferire con gli altri.

Sono quindi tre le problematiche da affrontare: l'autenticazione, l'autorizzazione e la riservatezza.

Per quanto riguarda le prime due, abbiamo già detto che un client può autenticarsi al sistema solamente se si è precedentemente registrato ed è autorizzato ad accedere al servizio se, al momento della connessione al server, inserisce la corretta coppia username/password. Per ciò che concerne l'accesso alla macchina virtuale via VNC, l'utente può accedervi solo conoscendo la password scelta in fase di registrazione al sistema.
La riservatezza, infine, è ottenuta grazie alla cifratura della comunicazione tramite un tunnel TLS, creato dal dispatcher quando si richiede il servizio, o generato dal nodo (tramite il protocollo VeNCrypt) quando ci si connette ad una macchina virtuale.

Capitolo 6

Implementazione della piattaforma

6.1 Ruby e OpenNebula

Come abbiamo già detto in precedenza, abbiamo scelto OpenNebula come piattaforma di cloud computing per la sua grande flessibilità, interoperabilità e semplicità d'uso.

Per quanto riguarda la codifica, abbiamo implementato il nostro progetto utilizzando il linguaggio Ruby, questa scelta è guidata da varie ragioni.
Innanzitutto, come abbiamo precedentemente visto, OpenNebula fornisce le API per Ruby. Il fatto di avere a disposizione queste API, rende l'infrastruttura molto più leggera ed il codice più leggibile, dato che non è necessario fare una chiamata ad una shell esterna al programma.

Ruby, inoltre, è un linguaggio interpretato e gode ormai di un'ampia diffusione. Esistono infatti interpreti per i più importanti sistemi desktop (come Linux, Windows, Mac OS X, etc...) e mobile (come Android, Symbian, etc...); per questo è quindi possibile riutilizzare il codice su tutti questi sistemi operativi senza dover apportare modifiche.
Questo è molto comodo in generale, ma diventa quasi essenziale in un progetto indirizzato ad un gran numero di utenti che utilizzano un insieme

eterogeneo di sistemi operativi.

Infine abbiamo scelto questo linguaggio anche per la sua semplicità ed immediatezza e, pur essendo un progetto relativamente giovane, dispone di un numero di librerie molto ampio.

In due casi, ovvero la registrazione degli utenti e il controllo del carico dei nodi, è stato necessario salvare alcuni dati in maniera più sicura e strutturata di un semplice file o variabile interna al programma; abbiamo quindi deciso di salvare i dati suddetti in due database SQLite [19]. Abbiamo scelto questa libreria perché è open source, estremamente leggera e veloce e, dovendo trattare pochi dati con semplici operazioni SQL, le limitazioni che presenta rispetto a database più completi come MySQL, sono ininfluenti.

6.2 Il dispatcher e i nodi

Come abbiamo visto nel capitolo precedente, il "centro di comando" della piattaforma è il dispatcher e i centri di calcolo sono i nodi della nuvola.
Si noti che nell'attuale implementazione, i nodi hanno il repository delle immagini delle macchine virtuali condiviso e il front-end, invece, è completamente separato (si veda lo schema in figura 6.1). Abbiamo optato per questa soluzione in quanto è maggiormente scalabile (rispetto al file system condiviso fig. 3.1) dato che, nel caso volessimo aggiungere un cluster di nodi, separato e distante da quello iniziale, la nostra infrastruttura non risentirebbe della latenza data dalla lontananza tra i nodi e il dispatcher.

Per impostare e rendere funzionante l'infrastruttura, l'amministratore di sistema deve aggiungere i nodi e le immagini delle macchine virtuali disponibili all'interno dell'ambiente di OpenNebula con i seguenti comandi:

```
$ onehost create host01 im_kvm vmm_kvm tm_shared dummy
$ oneimage create image.template
```

ed infine lanciare lo script Ruby "`ariaSrv.rb`" (vedi sorgente B.2).

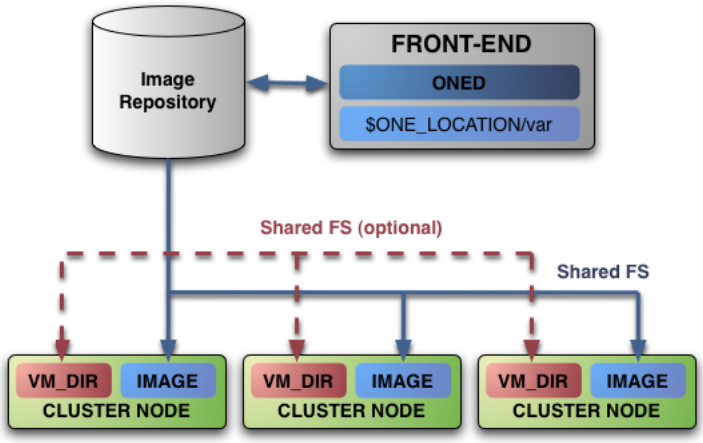

Figura 6.1: Schema dell'infrastruttura

Una volta fatto questo, sul server verrà eseguito un thread per monitorare le istanze delle macchine virtuali ed uno in attesa di richieste da parte dei client (un thread per ogni client).
Il thread di controllo è necessario per gestire le macchine virtuali in situazioni particolari. Se una macchina è in stato *unknown* o *failed*, significa che c'è stato un qualche errore nella virtualizzazione o che il nodo su cui era in esecuzione è caduto; in questi casi rimuoviamo semplicemente l'istanza dalla lista delle macchine virtuali.
C'è inoltre la possibilità che il client sia caduto prima di comunicare al dispatcher l'intenzione di disconnettersi; in questo caso dobbiamo controllare che tutte le macchine attive siano utilizzate da un client, se così non fosse, possono essere terminate, liberando così le risorse allocate.
Quest'ultimo controllo viene effettuato verificando che l'ultimo acknowledge ricevuto dal client non risalga a più di X minuti fa, dove X è un timeout predefinito (vedi sorgente B.1). Il client è tenuto, al fine di poter continuare ad usufruire del servizio, a contattare periodicamente il dispatcher, il quale salva il valore del momento di questo contatto nel database degli utenti, dove per ogni entry c'è il campo `ack`, preposto allo scopo suddetto.

Una volta ricevuta una richiesta da un client, viene lanciato un thread

dedicato e il dispatcher cerca il nodo migliore su cui eseguire la macchina virtuale desiderata; come suddetto, il concetto di "migliore" si basa su due unità di misura: i *bogomips* disponibili e la distanza dal client in termini di *latenza*.
Il primo valore viene calcolato da ogni nodo, al momento della richiesta da parte del dispatcher, in base al carico medio di CPU negli ultimi cinque minuti, moltiplicando la percentuale libera per i bogomips (sorgente B.3).
Dopo aver ottenuto i cinque nodi più liberi, il dispatcher invia questa lista al client che, tramite ping, calcola il secondo valore, identificando il nodo più vicino a se e restituisce al dispatcher la macchina selezionata.

Infine il front-end lancia la macchina virtuale richiesta sul nodo desiderato e comunica al client le porte vnc e udp relative alla macchina appena istanziata.

Sui nodi, invece, è in esecuzione un thread che ha il seguente scopo: inizialmente estrapola il valore dei bogomips totali e poi, ogni trenta secondi, calcola la percentuale di CPU libera lanciando le seguenti istruzioni:

```
$ cat /proc/info | grep bogo | awk {'sum+=$3;print sum'} | tail -n 1
$ ps aux | awk {'sum+=$3;print sum'} | tail -n 1
```

quest'ultimo dato viene salvato e si tiene traccia, in un database SQLite, degli ultimi dieci valori (per maggiori dettagli vedi sorgente B.4).
Quando il dispatcher lo richiede, viene fatta effettuata un'altra misurazione del carico della CPU e si calcola la media di questi undici valori, la quale viene utilizzata per estrapolare la percentuale di CPU libera; quest'ultimo dato si moltiplica infine per i bogomips totali, ottenendo la potenza di calcolo non utilizzata da altri processi.

6.3 Il client

Come già detto nel capitolo precedente, il client richiede il servizio al dispatcher che cerca un nodo opportuno su cui lanciare la macchina virtuale.

Nell'attuale implementazione sono distinti solamente due servizi: VoIP ed erogazione di macchine virtuali generiche. Questo significa che lo script client "`ariaCli.rb`" può essere lanciato con l'argomento *0* per richiedere il servizio VoIP (in questo caso sarà, di norma, lanciato da una terza applicazione), oppure con l'argomento *1* (per maggiori dettagli si veda il sorgente B.5).

Nel primo caso il server lancerà sul nodo opportuno il proxy server per il servizio VoIP; nel secondo caso, verranno mostrate all'utente le macchine virtuali disponibili tra cui poter scegliere (vedi figura 6.2).

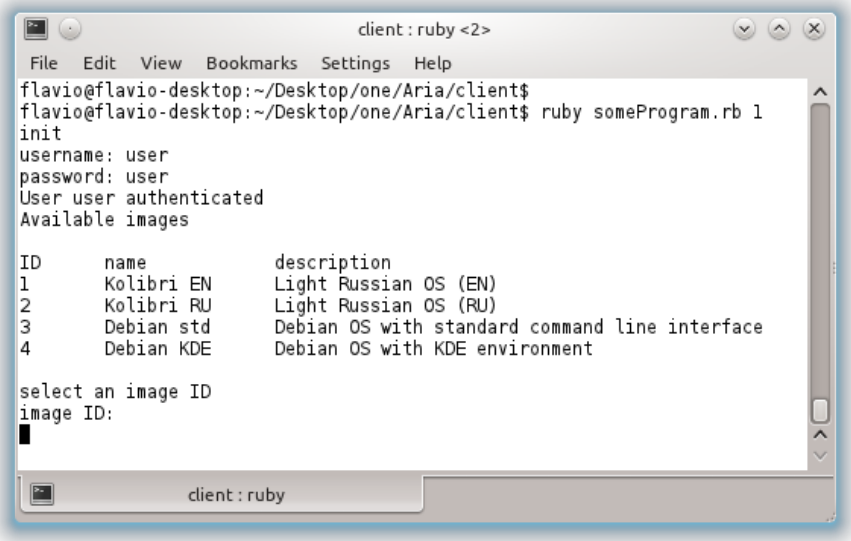

Figura 6.2: Schermata di selezione della macchina virtuale

Una volta selezionata l'immagine desiderata (o predefinita nel caso di servizio VoIP), il client lo comunica al front-end, che, come descritto nella sezione precedente, inizializza la macchina virtuale.

Il client, come abbiamo visto nella sezione precedente, una volta iniziata la connessione, e per tutto il periodo di utilizzo del servizio, deve periodicamente comunicare al server centrale di essere ancora attivo, per permettergli di distinguere gli utenti attivi da quelli che hanno terminato la sessione in modo

anomalo e di spegnere le macchine virtuali, eventualmente ancora attive, richieste da questi. Per espletare questo compito il client deve lanciare lo script con il parametro *2*.

Infine, in condizioni normali, il client comunica al dispatcher la volontà di finire la sessione, consentendo di terminare la macchina virtuale e liberare le risorse allocate. Questa operazione è compiuta eseguendo lo script client, passandogli come primo argomento *3* e come secondo parametro la lista o il range delle macchine virtuali da terminare o la stringa *all*, per comunicare la volontà di concludere la fruizione di tutti i servizi richiesti.

Capitolo 7

Test effettuati

Al fine di osservare il comportamento della piattaforma e verificarne il corretto funzionamento, abbiamo effettuato alcuni test.

Prima abbiamo effettuato i cosiddetti "performance test", utilizzati per indagare le performance di un programma, abbinati ai "load test", utilizzati per verificare il comportamento della piattaforma se sovraccaricata.
Poi abbiamo effettuato un test per verificare il supporto del multihoming.

7.1 Performance test

Per effettuare questi test abbiamo creato uno script che richiede al sistema l'erogazione di N macchine virtuali (N è un intero passatogli come argomento). In questo modo abbiamo potuto valutare se il sistema bilancia correttamente il carico di lavoro tra i vari nodi e, richiedendo contemporaneamente un numero elevato di macchine, se lavora correttamente anche sotto stress.

Nell'effettuare il test e le relative misurazioni, abbiamo eseguito tre volte il seguente procedimento, suddiviso a sua volta in tre step: prima abbiamo richiesto[1] l'erogazione di venti macchine, poi di altre venti, ed infine di quaranta ulteriori istanze del servizio, per un totale di ottanta macchine attive

[1]Le richieste sono state effettuate contemporaneamente da tre PC distinti

nel sistema contemporaneamente.
Dopo ogni set di richieste abbiamo misurato la distribuzione delle macchine virtuali, ovvero quante ne erano state lanciate su ogni nodo. Infine abbiamo calcolato, per ogni step, la media del numero di istanze in esecuzione su ogni host. Nell'appendice A sono riportate le tabelle relative ai risultati ottenuti dai tre test e la tabella delle medie.

Come si può vedere dalla tabella A.4, inizialmente le istanze delle macchine virtuali non sono ben bilanciate, ma questo era prevedibile ed auspicato, perché, dato che il sistema tiene conto soprattutto della distanza tra utente e nodi, il dispatcher tende a "sovraccaricare" il nodo più vicino all'utente. Richiedendo, però, un numero maggiore di macchine virtuali, il sistema si stabilizza e distribuisce le istanze in maniera sempre migliore.

7.2 Multihoming test

Questo test è stato necessario per verificare che gli utenti potessero effettivamente accedere ad una stessa macchina virtuale tramite diverse interfacce di rete, ognuna con un IP diverso.
Per simulare questa situazione ci siamo connessi ad una macchina virtuale da PC distinti, ognuno, ovviamente, con un IP diverso.

Conclusioni

La piattaforma da noi progettata, dovrebbe essere vista come un primo approccio e soluzione al problema che ci siamo posti inizialmente. Essa, infatti, sfrutta solamente una minima parte delle funzionalità offerte da OpenNebula ed è stata testata, per mancanza di risorse, su PC desktop sia come nodi sia come client.

Bisognerebbe, infatti, testare l'infrastruttura in un ambiente realistico, ovvero sarebbe opportuno avere a disposizione server dedicati con sistemi operativi orientati alla virtualizzazione e utilizzare, come immagini per le macchine virtuali, dei sistemi operativi configurati allo scopo di fornire connessioni VoIP; o ancora, utilizzare come dispositivi utente, dei veri terminali mobili, e creare applicazioni client dedicate al servizio, che sfruttino al meglio il multihoming ed il modello ABPS. Per esempio, si potrebbero utilizzare terminali mobili con sistema operativo Android, per il quale è disponibile un'implementazione del protocollo MIH di cui abbiamo parlato all'inizio.

Inoltre, sarebbe molto utile che in futuro si integri questa infrastruttura con servizi di cloud computing reali di tipo IaaS, tramite cui sarebbe possibile distribuire i nodi della nuvola su tutto il globo, migliorando il servizio e potendo usufruire di un parco macchine scalabile in base alle esigenze del momento. Ad esempio, essendo la nostra piattaforma basata su OpenNebula, sarebbe facilmente integrabile col servizio "Amazon EC2", per il quale, come abbiamo visto, OpenNebula offre le API necessarie.

Infine potrebbe essere molto utile ed interessante “misurare” il livello di sicurezza offerto dalla nostra piattaforma, ed eventualmente migliorarlo, al fine di offrire il servizio in massima sicurezza.

Appendice A

Tabelle dei test

Sono qui riportate le tabelle citate nella sezione 7.1 relative ai tre test effettuati. Nella prima colonna sono indicati i nomi dei server utilizzati per l'esperimento, nelle tre successive colonne è riportato il numero di macchine virtuali lanciate sul nodo per ognuno dei tre step del test (prima lancio di venti macchine, poi altre venti ed infine altre quaranta).

Nella quarta ed ultima tabella, abbiamo riportato la media delle macchine virtuali lanciate sui nodi in ogni step dei test e la relativa percentuale rispetto al totale delle istanze attive nel sistema al momento.

Nodo	Macchine virtuali in esecuzione per nodo		
abdallo	4	15	25
montano	6	7	18
radames	3	11	20
susanna	7	7	17
	20 macchine	40 macchine	80 macchine

Tabella A.1: Primo test

Nodo	Macchine virtuali in esecuzione per nodo		
abdallo	2	12	19
montano	12	12	22
radames	1	11	20
susanna	5	5	19
	20 macchine	40 macchine	80 macchine

Tabella A.2: Secondo test

Nodo	Macchine virtuali in esecuzione per nodo		
abdallo	11	11	20
montano	2	13	24
radames	6	6	17
susanna	1	10	19
	20 macchine	40 macchine	80 macchine

Tabella A.3: Terzo test

	Media macchine virtuali in esecuzione per nodo					
Nodo	Numero	Percentuale	Numero	Percentuale	Numero	Percentuale
abdallo	5,7	28,5%	12,7	31,75%	21,3	26,64%
montano	6,7	33,5%	10,7	26,75%	21,3	26,64%
radames	3,3	16,5%	9,3	23,25%	19,0	23,75%
susanna	4,3	21,5%	7,3	18,25%	18,3	22,87%
	20 macchine		40 macchine		80 macchine	

Tabella A.4: Media del numero di istanze per step per nodo

Appendice B

Sorgenti

```
173   def check_ack()
174     users_arr = Array.new
175     users = MySqLite::get_users
176     time_out = Time.at(TIME_OUT)
177     users.each do |user|
178       if !user[2].nil? && (Time.now - user[2]) > time_out
179         puts "#{user[0]} no ack removing vms.."
180         users_arr << user[0]
181         MySqLite::update_ack(user[0], nil)
182       end
183     end
184     return users_arr
185   end
```

Sorgente B.1: Funzione di controllo acknowledge Utils.rb

```
40 def main()
41
42   server = TCPServer.new(LIST_PORT)
43   sslServer = MySSL::set_contxt(server)
44   th_gr = ThreadGroup.new
45   semaphore = Mutex.new
46   Thread.abort_on_exception = true
47
48   th = Thread.start do
49     MyThread::thread_loop
50   end
51   th_gr.add(th)
52
53   loop do
54     begin
55       th = Thread.start(sslServer.accept) do |sslSocket|
56         puts "thread start"
57         MyThread::thread_routine(sslSocket, semaphore)
58       end
59     rescue => error
60       puts "error in thread: #{error}"
61       retry
62     end
63     puts "adding thread"
64     th_gr.add(th)
65   end
66
67 ensure
68   for th in th_gr.list
69     th.join(5)
70   end
71 end
```

Sorgente B.2: ariaSrv.rb

```
35 HOSTNAME = `hostname`.chomp + "." + `dnsdomainname`.chomp
36 db = Sequel.sqlite("stats.db")
37
38 vars = 1
39 cpu = `ps aux | awk {'sum+=$3;print sum'} | tail -n 1`.chomp.to_f
40 mem = `free -m | grep Mem | awk {'print $4'}`.chomp.to_i
41 cpus = db[:hosts].filter(:name => HOSTNAME)
42
43 # puts cpu
44
45 10.times do |i|
46   cpu_tmp = cpus.get(:"cpu#{i}")
47   if !cpu_tmp.nil?
48     cpu += cpu_tmp
49     vars += 1
50   end
51 end
```

Sorgente B.3: hostCheck.rb

```
35 HOSTNAME = `hostname`.chomp + "." + `dnsdomainname`.chomp
36
37 db = Sequel.sqlite("stats.db")
38 db.create_table?(:hosts){
39   String :name
40   Float :bogo,:default => nil
41   Float :cpu0,:default => nil
42   Float :cpu1,:default => nil
43   Float :cpu2,:default => nil
44   Float :cpu3,:default => nil
45   Float :cpu4,:default => nil
46   Float :cpu5,:default => nil
47   Float :cpu6,:default => nil
48   Float :cpu7,:default => nil
49   Float :cpu8,:default => nil
50   Float :cpu9,:default => nil
51   }
52
53 bogo = `cat /proc/cpuinfo | grep bogomips | awk {'sum+=$3;print sum'}
54         | tail -n 1`.chomp.to_f
55
56 db[:hosts].filter(:name => HOSTNAME).delete
57 db[:hosts].insert(:name => HOSTNAME)
58 db[:hosts].filter(:name => HOSTNAME).update(:bogo => bogo)
59
60 indx = 0
61 loop do
62
63   cpu = `ps aux | awk {'sum+=$3;print sum'} | tail -n 1`.chomp.to_f
64   db[:hosts].filter(:name => HOSTNAME).update(:"cpu#{indx}" => cpu)
65   indx = (indx+1)%10
66    sleep 30
67
68 end
```

Sorgente B.4: hostLoop.rb

```
42 class AriaCli
43
44   attr_reader :req, :opts, :username, :passwd
45
46   def initialize(req, opts=nil, username=nil, passwd=nil)
47     if !req.nil?
48       req = req
49       if !(req == 0 || req == 1 || req == 2 || (req == 3 && !opts.nil?))
50         Utils::usage
51       end
52     else
53       Utils::usage
54     end
55
56     @req      = req
57     @opts     = opts
58     @username = username
59     @passwd   = passwd
60   end
```

Sorgente B.5: Classe AriaCli (istanziazione) in ariaCli.rb

```
62   def main
63
64     socket = TCPSocket.new(SERVER, PORT)
65     sslContext = OpenSSL::SSL::SSLContext.new
66     sslContext.ca_file = "#{CERT_PATH}ca_cert.pem"
67     sslContext.verify_mode = OpenSSL::SSL::VERIFY_PEER
68
69     begin
70       ssl = Utils::MySSLSocket.new(socket, sslContext)
71       ssl.connect
72     rescue => error
73       puts "socket error: #{error}"
74     end
75
76     Utils::authenticate(ssl, @username, @passwd)
77
78     ssl.puts req
79     case req
80     when 0
81       Utils::req_voip(ssl)
82     when 1
83       Utils::req_vm(ssl)
84     when 2
85       Utils::req_ack(ssl)
86     when 3
87       Utils::req_close(ssl, @opts)
88     end
89
90   ensure
91     if !ssl.nil?
92       ssl.close
93     end
94   end
95 end
```

Sorgente B.6: Classe AriaCli (metodo principale) in ariaCli.rb

Bibliografia

[1] Amazon, *Amazon Elastic Compute Cloud*, `http://aws.amazon.com/ec2/`, 2012

[2] Bellard F., QEMU, Open source processor emulator, `http://wiki.qemu.org/`

[3] Caressa P., *Cos'è la Crittografia?*, `http://www.caressa.it/pdf/crypto.pdf`, 2005

[4] Dierks T., Rescorla E., *The Transport Layer Security (TLS) Protocol, Version 1.1*, `http://tools.ietf.org/html/rfc4346`, 2006

[5] Ferretti S., Ghini V., Panzieri F. (2011) "Structuring Clouds for Mobile Multimedia", IEEE COMSOC MMTC E-Letter, 6, 3, pp. 28-30

[6] Google Inc., *Google App Engine*, `http://code.google.com/intl/it/appengine/`, 2010, agg. 2012

[7] IEEE Group, *IEEE 802.21*, `http://www.ieee802.org/21/`

[8] Johnson B., *Cloud computing is a trap, warns GNU founder Richard Stallman*, `http://www.guardian.co.uk/technology/2008/sep/29/cloud.computing.richard.stallman`, 2008

[9] Red Hat, Inc., *The Kernel Based Virtual Machine*, `http://www.linux-kvm.org/`

[10] Red Hat, Inc., *Libvirt, the virtualization API*, `http://libvirt.org/`

[11] Mell P., Grance T., *The NIST Definition of Cloud Computing*, `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`, 2011

[12] OpenBSD, *OpenSSH Manual pages*, `http://www.openssh.org/manual.html`, 2012

[13] OpenCrowd, *Cloud Taxonomy*, `http://cloudtaxonomy.opencrowd.com/`, 2010

[14] OpenNebula Project Leads, *OpenNebula v3.2 Documentation*, `http://opennebula.org/documentation:rel3.2`, 2002, agg. 2012

[15] Pettey C., Stevens H., *Gartner Says Cloud Computing Will Be As Influential As E-business*, `http://www.gartner.com/it/page.jsp?id=707508`, 2008

[16] Richardson T., *The RFB Protocol v3.8*, `http://www.realvnc.com/docs/rfbproto.pdf`, RealVNC Ltd, 2010

[17] Ruby Community, *Ruby, A Programmer's Best Friend*, `http://www.ruby-lang.org/en/`, 2012

[18] Schubert L., *The Future of Cloud Computing*, `http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf`, 2010

[19] SQLite Community, *SQLite Documentation*, `http://www.sqlite.org/docs.html`, 2012

[20] The GNOME Project, *GTK-VNC - a VNC viewer widget for GTK*, `http://live.gnome.org/gtk-vnc`, 2005, agg. 2011

[21] Tilotta V., *Progettazione ed implemetazione di una piattaforma di cloud computing per l'erogazione di macchine virtuali*, Bologna, 2011

[22] VeNCrypt Community, *The VeNCrypt project*, `http://vencrypt.sourceforge.net/`, 2006, agg. 2012

[23] Wim van Dorst, *BogoMips mini-Howto, v.38*, `http://www.clifton.nl/index.html?bogomips.html`, 2006