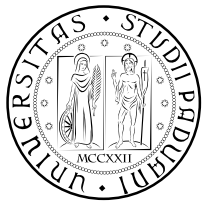

UNIVERSITÀ DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Misure di QoS nei sistemi Cloud: Stato dell'Arte

Relatore

Prof. **Carlo Ferrari**

Laureando

Ludovik Çoba

Matricola 607286-IF

Anno Accademico 2012-2013

"Ahimè!, ho studiato, a fondo e con ardente zelo,
filosofia e giurisprudenza e medicina e, purtroppo, anche teologia.
Eccomi qua, povero pazzo, e ne so quanto prima! "
GOETHE, JOHANN WOLFGANG (1749-1832)

"Good news, everyone! "
HUBERT J. FARNSWORTH

Abstract

The Cloud Computing is a hot topic on the IT. It is videly used by everyone and it rapresents an important infrastructure for the community. But there are still many solution to be developed on it, like the QoS.

In this thesis was presented the Quality of Services for the cloud, it was fragmented and exposed some problems regarding it. Over each sub-topic are introduced some frameworks. The frameworks are explained , discussed, commented and highligted some problems regarding them.

Sommario

Il Cloud computing è un topico rovente per quanto riguarda l'ICT. Esso è relativamente nuovo in continuo sviluppo e offre servizi per tutti. Però nonostante questo tanti ambiti sono ancora da sviluppare, coma la QoS.
In questa tesi si è esplorato il topico della qualità dei servizi, scompattato e visti alcuni problemi che riguardano questi frammenti. Per ogni sottotopico si presentano alcuni modelli relativi. I modelli vengono spiegati e argomentati aggiungendo per ciascuno delle ideo e sottolineando alcuni problemi che nascono durante la loro applicazione.

Indice

Elenco delle figure

Introduzione

Il Cloud Computing rappresenta un topico rovente del Communication and Information Tecnology. Esso si e espanso a macchia d'olio entrando nella vita sia delle persone che applicandosi all'industria. Il cloud è cosi importante perchè offre uno svariato numero di vantaggi come elevata quantità di risorse di qualsiasi tipo, con una struttura pay-as-you-go e, inoltre, disassocia l'utente dai costi elevati di manutenzione dell'hardware. Questa tecnologia viene distribuita sotto forma di servizio e di conseguenza nasce spontaneo parlare di qualità dei servizi (QoS). Però essendo la tecnologia relativamente giovane il contributo su questo argomento è ancora piccolo e non tutti gli aspetti della QoS sono stati ricoperti.

In questa tesi si sono osservati gli aspetti fodamentali che caratterizzano la qualità dei servizi per il cloud. Inoltre si è visto la maniera con la quale si tenta di rispondere alle richieste di qualità e sono stati argomentati alcuni approcci. Questo lavoro è presentato come segue.

Nel primo capitolo viene data una ottica del cloud computing e acluni aspetti base, con alcune nozioni che sono fondamentali per comprendere come il cloud funzione. Si inquadrono le principali caratteristiche fondamentali sulla distribuzione e utilizzo del cloud. Inoltre si spezza la QoS in cinque aspetti principali e si definisce il SLA.

Nel resto del lavoro si presentano i cinque aspetti individati come fondamentali per la QoS. Gli aspetti definiti sono *le prestazioni, la scalabilià, disponibilità, affidabilità e sicurezza.* La loro definizione viene fatta in un ottica di problematiche e

argomenti da tenere sottocchio per la loro determinazione e astpetti da trattare per offire servizi qualitatevoli. Gli attributi di QoS vengono analizzati separatamente per determinare l'eventuale atomicità di essi. Successivamente per ogni attributo si presentano alcuni schemi peculiari che trattano e tentano di sistemare l'attributo per garantire un livello minimo di qualità dei servizi.

Capitolo 1

Cloud Compting

1.1 Definizione

Il *Cloud Computing* è un paradigma assiociato all'approvigionamento di infrastrutture computazionali. Questa l'infrastruttura si colloca sul web con lo scopo di ridurre costi assiociati alla gestione di risorse harware e software [1]. Il topico sul Cloud Computing esplose nel tardo 2007, e attirò l'attenzione per il fatto che offriva infrastrutture dinamiche per l'IT e servizi software facilmente configurabili. In Figura 1.1 si può vedere come il Cloud ha superato e sostituito il Grid Computing [3].

Figura 1.1: Google Trend, confronto tra Grid e Cloud Computing

Al momento il Cloud Computing ha invaso sia settori industriali che quelli accademici, si può citare: Reservoir Project [4], Amazon EC2 [5], OpenNebula[6] e progetti scentifici come Nimbus [7] .

Nonostante la sua diffusione non esiste ancora una definizione formale sul Cloud.

Secondo [8] quella piú citata nelle bibliografie è quella del National Institute of Standards and Technology (NIST)[2] che afferma che il Cloud Computing è un modello che offre comodità, ubiquotà, accesso on-demand ad una sorgente condivisa di risorse computazionali(come: server, reti, memoria, applicazioni e servizi) l'approvigionamento dalle quali si possa compiero con il minimo sforzo di gestione o la minima interazione del fornitore del servizio. Un modello simile di cloud deve essere composto cinque caratteristiche essenziali, tre modelli si servizio e quattro metodi di distribuzone.

1.2 Caratteristiche Cloud

Per parlare dei benefici che porta l'utilizzo di un modello cloud riprendiamo la definizione di Cloud Computing del NIST nella quale sono descritte le cinque principali caratteristiche offerte [2]:

- *Self-service su richiesta.* L'utente può unilateralmente approvigionare le risorse per calcoli ed elaborazioni(server e spazi di memoria). Tale operazione deve avvenire in maniera automatica senza richiedere interventi da parte del fornitore o interazioni umane.

- *Ampia possibilità di accesso.* Le risorse sono disponibili sulla rete. Esse vengono accedute tramite meccanismi standard e sono utilizzabili da una larga gamma di piattaforme (ad esempio, telefoni cellulari, tablet, computer portatili etc.).

- *Pooling di risorse.* Le risorse di calcolo sono organizzate in modo da servire numerosi consumatori tramite modelli multi-tenant. Le risorse virtuali e fisiche vengono distribuite e ridistribuite in base alle richieste del consumatore(scalabilità [17]). L'utente viene colto da una senzazione di distacco e impotenza per quanto riguarda la collocazione e il controllo delle risorse in usufrutto. Lútente comunque può conoscere la locazione ad un alto livello di astrazione(come stato, paese o centro dati).

- *Elasticità.* Le risorse possono essere utilizzate e cedute in maniera elastica, in alcuni casi anche in maniera automatica[1], per ridimensionare lútilizzo in base alle richieste dell'utente. Spesso, all'utenti si crea l'idea che le risorse che egli può utilizzare siano infinite e disponibili tutto il tempo.

- *Servizi misurati.* I sistemi Cloud controllano e ottimizzano l'utilizzo delle risorse con una misura di dosaggio appropriata in base al tipo di servizio utilizzato. L'utilizzo delle risorse può essere monitorato, controllato, offrendo cosí trasparenza sia all'utente che al fornitore.

[1]Si intende i casi nei quali l'utente ha richiesto e pagato per piú risorse di quelle utilizzate, in questo caso l'eccesso viene distribuito. Nel caso l'utente avesse necessità delle risorse in piú, può accederle immediatamente.

Inoltre sempre con riferimento al Cloud Computing possiamo aggiungere i seguenti elementi:

- *Costi Ridotti.* L'alta efficienza delle reti cloud porta ad una forte riduzione dei costi.
- *Qualità di servizio.* QoS è uno delgi aspetti principali nei sistemi orientati ai servizi. La QoS fornisce un livello di certezza che le risorse necessarie ad una applicazione sono rigorosamente raggiunte.
- *Affidabilità.* L'affidabilità viene garantita dall'ottima capacità di bilanciamento del carico nelle reti cloud superiore a quello garantito da un singolo organismo.
- *Gestione dell'IT fuori sede.* La dislocazione del cloud computing permette di concentrare l'attenzione solo sul business dell'azienda, lasciando la gestione IT ad altri con conseguente risparmio sui costi.
- *Manutenzione ed aggiornamenti semplificati.* Gli utenti hanno accesso a strutture cloud sempre aggiornati, grazie al sistema centralizzato offerto dal cloud.

1.3 Architettura cloud

E' possibile descrivere l'architettura di un ambiente di cloud computing modellizzandola in quattro strati: hardware layer, infrastructure layer, platform layer e application layer. Segue la descrizione di ciascuno di questi livelli.

- Hardware layer: questo strato è responsabile della gestione delle risorse fisiche del cloud, incluse le macchine fisiche (server), i routers, gli switches e i sistemi di alimentazione e raffreddamento. Lo strato hardware 'e tipicamente implementato nei data centers. Un data center contiene migliaia di server che sono organizzati in racks. La gestione hardware comprende operazioni di configurazione, realizzazione di infrastrutture fault-tolerant, gestione del traffico di rete e gestione delle risorse.

- Infrastructure layer: chiamato anche strato di virtualizzazione, questo strato fornisce le risorse fondamentali agli strati di pi‘u alto livello, generalmente servizi computazionali, data storage e servizi di rete, attraverso la creazione di pool di memoria e risorse computazionali partizionando le risorse fisiche attraverso tecnologie di virtualizzazione come Xen, KVM and VMware . L'infrastructure layer ‘e un componente essenziale del cloud computing, perch´e molte caratteristiche chiave, come l'assegnamento dinamico delle risorse, sono disponibili soltanto attraverso il meccanismo di virtualizzazione.

- Platform layer: gli utenti di questo strato sono gli sviluppatori, che implementano le loro applicazioni e ne fanno il deploying sulla cloud. Gli infrastructure providers supportano queste operazioni fornendo dei framework di sviluppo che consistono in insiemi di API per facilitare l'interazione con l'ambiente cloud o il cui scopo ‘e quello di minimizzare il peso del deploy delle applicazioni all'interno delle macchine virtuali. Ne sono un esempio i servizi di auto-scaling e load-balancing offerti da Amazon, che si occupano di ottimizzare l'allocazione dinamica delle istanze virtuali in modo trasparente alle applicazioni. In generale gli sviluppatori hanno la possibilit‘a di integrare on-demand questi servizi nelle loro applicazioni.

- Application layer: questo ‘e lo strato con cui interagiscono gli utilizzatori finali, i quali accedono ai servizi offerti attraverso il web. Il deploy di una cloud application viene eseguito sull'infrastruttura offerta da un infrastructure provider in modo che gli sviluppatori possano apportare modifiche al software o aggiungere nuove funzionalit‘a senza che si renda necessaria la distribuzione di update o service pack. A questo livello diventano critici gli aspetti di sicurezza e di qualit‘a del servizio, che vengono regolati attraverso i contratti di Service Level Agreement (SLA).

Rispetto ai tradizionali servizi di hosting, come le server farms, l'architettura del cloud computing ‘e modulare. In modo simile al modello OSI dei protocolli di rete, ciascun livello ha basso accoppiamento con quelli ad esso adiacenti, permettendo a ogni strato di evolvere separatamente.

Il Cloud computing offre i suoi servizi in accordo a tre modelli fondamentali[18][2](figura 1.2): Software as a Service(SaaS), Platform as a Service(PaaS) e Infrastructure as a Service(IaaS) i quali verranno spiegati nelle tre sottosezioni che seguono.

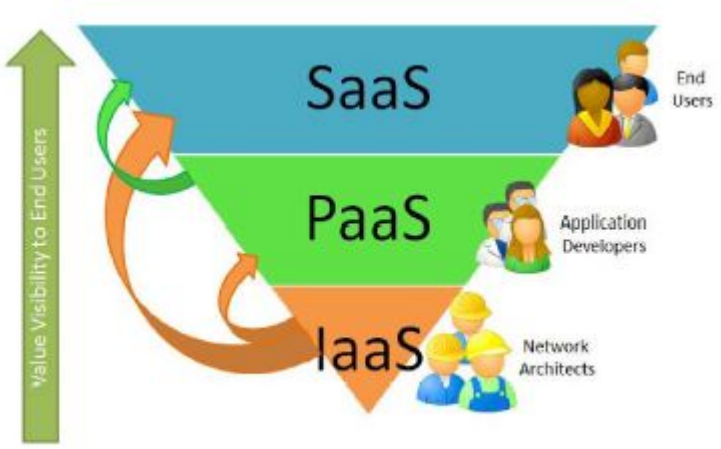

Figura 1.2: Modello Cloud.

1.3.1 Software as a Service(SaaS)

Con Software as a Service (SaaS), all'inizio noto anche come Software on Demand (SoD), intendiamo la possibilità di usare applicazioni, in modalità centralizzata, in esecuzione su una cloud infrastructure, accessibile da vari dispositivi client attraverso una interfaccia, come un browser web.
Il consumatore vede solo il risultato finale, e cioè l'applicazione, non riesce a controllare l'infrastruttura di base (rete, server, sistemi operativi, storage). L'utente potrà accedere e modificare solo specifiche impostazioni, o aggiungere funzionalità opzionali, durante la sessione di configurazione. Dividiamo il servizio SaaS in due categorie:

- Linea di Servizi alle Imprese: si riferiscono a soluzioni di business offerte alle imprese e venduti o messi a disposizione di queste sulla base di un abbonamento
- Orientato al Cliente: sono servizi che vengono offerti al pubblico in generale su una base di sottoscrizione o, più spesso, offerti gratuitamente e supportati

dalla pubblicità (sono di questa categoria servizi come il pacchetto Windows Live di Microsoft e Google Docs).

I software offerti sono condivisi tra tutti i clienti e garantisce la separazione logica dei dati tra utenti differenti. L'affdabilità sulla integrità e sul salvataggio dei dati è una caratteristica fondamentale, e sarà compito dei data centers interni al provider assicurare la replicazione e il backup, l'utente non deve preoccuparsi di queste operazioni.
Le applicazioni on-demand seguiranno la logica del "pay-as-you-go venendo pagate solo per l'effettivo utilizzo (in ore o in carico computazionale), in questo modo l'utente sarà sollevato dal pagare licenze e preoccuparsi di manutenzioni. Il risultato comporta un notevole risparmio, soprattutto per le aziende che usufruiscono di servizi a pagamento.
Per alcuni Web Services è possibile l'integrazione con altri software avendo così la possibilità di sviluppare nuove applicazione di tipo SOA (Software Oriented Architecture).

1.3.2 Platform as a Service (PaaS)

I servizi di Platform as a Service (o PaaS) forniscono al consumatore la possibilità di distribuire nella Cloud Infrastructure applicazioni create dall'utente che utilizzano linguaggi di programmazione supportati dal fornitore. Rispetto al SaaS il consumatore ha il controllo sulle applicazioni distribuite ed eventualmente sulle configurazioni dell'ambiente.
Sarà compito del provider incaricarsi delle decisioni riguardanti l'ambiente in cui il software verrà sviluppato ed eseguito, del sistema operativo messo a disposizione, il linguaggio di programmazione e le relative API's, le varie configurazioni della piattaforma. L'utente è esonerato da tali compiti, la piattaforma infatti, tramite la virtualizzazione, permette di svilupparvi applicazioni all'interno senza preoccuparsi della struttura hardware sottostante.
Esempi di PaaS sono SalesForce.com, Windows Azure, Joyent (Public Cloud utilizzata anche da Facebook e LinkedIn) e Google Apps.

1.3.3 Infrastructure as a Service (Iaas)

L'Infrastructure as a Service (o IaaS), detto anche Hardware as a Service, d a al consumatore la possibilità di poter noleggiare capacità di CPU, storage, network e altre risorse fondamentali che l'utente è in grado di implementare e gestire, possono includere i sistemi operativi e le applicazioni. Il consumatore ha il controllo su sistemi operativi, storage ecc.. e seleziona i componenti di rete (Load Balancer, Firewall); il cliente non ha il controllo delle infrastrutture di base della Cloud.
I servizi messi a disposizione si possono dividere in tre categorie:

- Dispositivi: server aziendali, dispositivi di storage, di rete e sicurezza.
- Impianti e strutture: alloggiamenti per dispositivi, datacenters, sistemi di raffreddamento, generatori di potenza e sistemi di backup e sicurezza.
- Sistemi di gestione: monitoraggio delle prestazioni (on-site o da remoto) e modifica delle impostazioni in caso di problemi. Lo IaaS crea, tramite la virtualizzazione, una versione virtuale di una risorsa normalmente fornita fisicamente. Nonostante il grande numero di possibilità che l'infrastruttura offre, rimane compito del provider l'acquisto, la manutenzione e la gestione delle risorse hardware; l'utente si preoccuperà solo dell'acquisto di CPU, storage, larghezza di banda necessarie per l'esecuzione e lo sviluppo delle proprie applicazioni. Il modello tariffario applicato è sempre quello del ''pay-as-you-go come negli altri modelli.

Esempi di Infrastructure as a Service sono Icloud (Storage ed applicazioni accessibili da browser o cellulare), MobileMe (Cloud Computing progettato per iPhone ed iPad da Apple), Rackspace (soluzioni di Private Cloud), VMWare (soluzioni per Paas e Iaas), Amazon Web Services (con EC2 per l'acquisto di capacità di calcolo e S3 per lo storage come servizi principali, sia come Paas che come IaaS).

1.4 Modelli di distribuzione

- *Cloud pubblico.* L'infrastruttura cloud è resa disponibile a chiunque e a qualsiasi organizzazione. Gli enti che utilizzano questo modello pagano i servizi utilizzati oppure li ottengono gratuitamente, l'accesso è effettutao tramite internet. Il vantaggio di una rete pubblica sta tra una semplice gestione, che non prevede i costi dell'hardware, applicazioni e costi di banda sono a carico del provider, e la scalabilià del sistema con pochi limiti e migliore gestione delle risorse.

- *Cloud comunitaria.* Infrastruttura claud scpecifica ad enti o gruppi che condividono obbiettivi e necessità (sicurezza, missioni etc.).

- *Cloud privato.* Infrastruttura cloud accesibile soltanto all'interno di una rete privata. Utilizzata imprese o terze parti che offrono questo servizio alle imprese. Offre servizi di hosting ad un numero limitato di utenti dietro ad un firewall. I progressi nella virtualizzazione e il calcolo distribuito hanno permesso agli amministratori della rete aziendale e dei data center di diventare effettivamente fornitori di servizi che soddisfano le esigenze dei loro "clienti" all'interno della società.

- *Cloud hibrido.* Infrastruttura cloud composta da piú distribuzioni distinte di cloud(privata, pubblica o comunitaria). Può offrire accesso a dati o ad applicazioni standardizzate o proprietarie. Viene generalmente offerta con due soluzioni: un fornitore dispone di un cloud privato e costituisce una partnership con un provider che gestisce una cloud pubblica, o viceversa, un provider che gestisce cloud pubbliche costituisce una partnership con un fornitore che fornisce piattaforme su cloud private.

  Una Cloud Hibrida è un ambiente di cloud computing in cui l'organizzazione fornisce e gestisce alcune risorse in-house mentre altre vengono fornite esternamente. Ad esempio, un'organizzazione potrebbe utilizzare un servizio di cloud pubblico, come Amazon S3 per i dati archiviati continuando per'o a mantenere in-house i dati di archiviazione dei clienti. Idealmente, l'approccio ibrido consente alle aziende di trarre vantaggio dalla scalabilità e dal rapporto

costo-beneficio che un'ambiente di cloud computing pubblico offre senza esporre applicazioni critiche e ati a vulnerabilit'a di terze parti.

1.5 QoS e SLA

L'interesse per applicazioni Internet è in costante crescita. Alcuni servizi esistenti e emergenti richiedono elevati livelli di qualità del servizio (quality of service o QoS) e hanno elevate esigenze di risorse (si pensa ad applicazioni real-time come videoconferenze). In letteratura la Qualità del Servizio (QoS) non trova una definizione univoca bensì differenti definizioni a seconda del campo a cui è applicata. Troviamo una definizione di qualità del servizio come l'effetto collettivo delle performance di un servizio che determinano il grado di soddisfazione di un utente del servizio [16]. Quando un'azienda o organizzazione si affida a servizi forniti da un altro ente o impresa (per esempio servizi web) per l'implementazione dei propri processi business, spesso vengono richieste garanzie contrattuali sulla qualità del servizio, come allo stesso modo i fornitori del servizio richiedono garanzie affinché i clienti non abusino del servizio. Queste qualità e i vincoli di utilizzo sono spesso definiti in accordi bilaterali chiamati Service Level Agreement (SLA), che specificano la qualità del servizio richiesta e le penalità associate alle violazioni. Queste penalità sono tradotte con pagamenti pecuniari (o rimborso per il costo del servizio) e possono essere viste come un'assicurazione contro la fornitura di un servizio scadente e l'eccessivo utilizzo. Le applicazioni enterprise, a differenza delle normali applicazioni, hanno stringenti esigenze di qualità del servizio. Con ciò si intende una serie di caratteristiche, tipicamente requisiti non funzionali come disponibilità, scalabilità, affidabilità e tempistiche di risposta, che l'applicazione deve rispettare e che solitamente sono espresse attraverso l'uso di contratti SLA, che legano un fornitore di servizi ad un cliente che ne fa uso. I contratti SLA possono essere di diversi tipi e con diversi scopi. Attualmente la maggior parte dei cloud provider (pubblici) offre SLA limitati alla disponibilità del servizio e/o alle caratteristiche delle risorse offerte (es. numero e tipo di CPU).

Testare la qualità dei servizi web utilizzando, per esempio, test su performance e affidabilità è necessario, ma non sufficiente. La qualità del servizio dipende

fondamentalmente dalla fornitura di risorse computazionali che il service provider (o chi per esso) gestisce durante la vita del servizio. Per vigilare su una SLA, è necessario per l'utente del servizio monitorare costantemente, o almeno su intervalli statisticamente significativi, la qualità del servizio fornita a tempo di esecuzione. Inoltre il service provider dovrà anch'esso monitorare a run-time la qualità del servizio per controllare che l'utilizzo del servizio non ecceda i livelli concordati nella SLA, per proteggersi contro false affermazioni sulla scarsa qualità del servizio, ma soprattutto per determinare se aumentare le quantità di risorse utilizzate in caso la qualità del servizio si abbassa al di sotto di determinate soglie.

Molte applicazioni distribuite di classe enterprise vengono sviluppate per essere eseguite su piattaforme di Application Server, come J2EE, CORBA o .NET. Queste applicazioni possono richiedere requisiti di Qualità del Servizio (QoS), come scalabilità e disponibilità del servizio, attraverso un contratto di Service Level Agreement (SLA). Le SLA sono contratti che stabiliscono le garanzie di QoS che un ambiente di esecuzione deve fornire per le applicazioni che ospita. Per assicurare che la SLA di una applicazione non sia violata, una delle politiche adottabili è quella chiamata resource over-provisioning: vengono allocate staticamente un numero sufficiente di risorse per supportare un carico di lavoro in qualsiasi scenario, anche nel caso peggiore. Con questa politica però, una percentuale elevata di risorse può rimanere inutilizzata per gran parte del tempo. Al contrario una politica di utilizzo ottimale delle risorse può essere ottenuta fornendo a ogni applicazione ospitata il numero minimo di risorse richieste per onorare la SLA e continuando a gestire l'allocazione delle risorse anche a tempo di esecuzione.

I contratti di Service Level Agreement (SLA) sono l'attuale pratica del mondo economico per specificare requisiti di qualità del servizio nell'ambito IT. La SLA in rappresenta una collezione di clausole contrattuali che legano un QoSaware cluster alle applicazioni che ospita. Questo particolare tipo di SLA (scritta utilizzando il linguaggio SLAng [44]) prende il nome di hosting SLA. Questa può comprendere due macro aree: diritti e obblighi dei client (Client Responsabilities) e diritti e obblighi della parte server (Server Responsabilities). Tra gli obblighi dei client troviamo il numero massimo di richieste che i client possono inviare all'applicazione entro un

certo intervallo di tempo.
Il frammento riportato in figura 1.3 mostra l'attributo requestRate che serve appunto per esprimere il numero massimo di richieste (100) che è possibile inviare alla applicazione in un secondo. Gli obblighi dell'applicazione possono includeregaranzie su disponibilità del servizio, latenza delle risposte e percentuale delle violazioni sui termini della SLA che possono essere tollerate.

```
<ContainerServiceUsage name="HighPrority" requestRate="100/s">
            <Operations>
                <Operation path="catalog.jsp" />
                <Operation path="AddToCart" />
                <Operation path="checkout.jsp" />
                <Operation path="CheckoutCtl" />
            </Operations>
            ...
        </ContainerServiceUsage>
```

Figura 1.3: Frammento di codice SLA.

Un ulteriore parametro che si può notare negli obblighi della parte server è il maxResponseTime che rappresenta il tempo massimo che deve trascorrere tra la ricezione di richiesta (per una delle operazioni specificate all'interno del tag Operations) e il completamento dell'invio della relativa risposta. Infine per specificare la percentuale di violazioni della SLA che possono essere tollerate prima che un application service provider incorra in penalità, sono stati utilizzati i parametri efficiency e efficiencyValidity. Nell'esempio questi attributi specificano che non meno del 95% delle richieste, in un intervallo di due ore, deve essere soddisfatto in conformità ai requisiti specificati nella parte di Server Responsabilities.
Nel Cloud Computing emerge un insieme questioni sulla gestione dell'IT, e l'utilizzo di SLA nel Cloud è la soluzione per garantire un livello accettabile di QoS. Secondo [42] il SLA deve percorrere quattro passi: definizione di contratto SLA, negoziazione SLA, monitoraggio SLA e l'applicazione SLA. La fase di monitoraggio e l'applicazione del SLA è fondamentale per costruire un rapporto di fiducia tra il distributore del servizio e il cliente. Per applicare il SLA in un ambiente dinamico, come il Cloud, è necessario monitorare gli attributi QoS continuamente.
Il cloud e' una sorgente di un grande numero di risorse virtuali, intese come servizi

e distribuite su richiesta. Le risorse nel cloud vengono suddivise in queste categorie: IaaS, PaaS, SaaS. Pertanto il SLA puo‘ essere suddiviso in queste tre categorie [43]. Il SLA è influenzata dalla performance di rete, gli standard di qualità del servizio, le prestazioni delle applicazioni create dagli utenti e configurazione dei parametri. La definizione di un SLA per un servizio di cloud computing ha bisogno di prendere in considerazione questi problemi e, in definitiva, si può ottenere un accordo in grado di controllare il livello di rischio e fare in modo che i servizi di cloud computing si raggiungano gli obiettivi aziendali prefissati.

1.6 Voci del QoS per il cloud

Ogni attività e funzionalità del Cloud Computing viene distribuita all'utente come servizio, ciò rende inevitabile discutere la qualità dei servizi. In ogni sistema orientato ai servizi le principali proprietà non funzionali che sono percepibili dal cliente e riguardano la QoS sono *l'affidabilità, sicurezza e performance.* L'affidabilità e la sicurezza sono due aspetti del cloud che sono strettamente collegati tra di loro. Lo spostarsi dai personal computer verso il cloud computing è influenzata da una serie di fattori, l'affidabilità e sicurezza sono tra i fattori fondamentali(infatti l'utente è intimorito dal fatto che la piattaforma non è più sotto il suo controllo). Invece i problemi riguardanti le prestazioni si possono percepire come tempo di risposta, throughput e utilizzo rete.
Altro aspetto importante che riguarda la QoS su tutti i sitemi distribuite e anche il cloud è la *disponibilità delle risorse.* La *scalabilità* è un altro attributo di QoS che è particolarmente difficile da raggiungere, soprattutto a causa della natura sconosciuta dei consumatori di servizi e volumi imprevedibili di chiamate di servizio.

Capitolo 2

Prestazioni

Prima della nascita del cloud computing, uno dei compiti più difficili per gli sviluppatori di applicazioni web era la pianificazione delle strutture di calcolo. Ai fini di garantire le prestazioni delle applicazioni web, c'è bisogno di determinare la quantità di risorse di calcolo (ad esempio, CPU, memoria) e le che necessità devono essere acquistate in base ai requisiti per lo sviluppo del sofware, inoltre si deve ottenere una stima del carico futuro. I requisiti del software possono essere capiti con un analisi, collaudo e monitoraggio, ma il carico futuro è difficile da determinare a causa della sua imprevedibilità e variabilità, che in tal modo a volte porta ad avere una brutta preformance dell'applicazioni web. Nel cloud computing, gli sviluppatori non hanno più bisogno di fare la pianificazione delle risorse. Essi acquistano risorsa limitate da un provider di cloud in base al variare del carico di lavoro, e sottoscrivono un SLA. Poi l'approvigionamento delle risorse e la manipolazione viene passato al provider di cloud ed è trasparente agli sviluppatori.

Tuttavia, a causa della complessa architettura del cloud e l'interazione tra le varie applicazioni che condividono risorse di calcolo, è molto difficile far rispettare il SLA di ogni applicazione. L'esperimento condotto in [19] mette a confronto le prestazioni di applicazioni distribuite in MapReduce, Amazon EC2 e in un cluster locale, e il risultato dimostra che MapReduce soffre di varizioni in prestazioni notevolmente più elevate che in EC2.

Per il cloud raggiungere e mantenere il SLA per tutte le svariete richieste da parte degli utenti è un problema. Un approccio comune è quello di monitorare il

funzionamento dell'intero cloud, raccogliere dati di runtime, e adeguando il carico delle risorse manualmente[20]. Però nascono due problemi da affrontare. Il primo è che i dati ottenuti dal monitoraggio in runtime in ambiente cloud sono troppo complessi da comprendere e da analisare per l'attore uomo. Il secondo è la difficolta di effettuare una distribuzione ottimale per tutte le applicazioni. Altre misurazioni ottenuti dai test recenti su alcuni cloud commerciali [21], come Amazon EC2, indicano che le prestazioni della rete hanno un impatto significativo sulla qualità di servizio del cloud. Ciò rende la performance uno degli attribuiti fondamentali di un servizio per determinare la QoS.

Nelle sezioni successive verranno introdotti e commentati alcuni modelli di rilevanza che trattano alcuni aspetti fondamentali per garantire le prestazioni come elemento della QoS.

2.1 Prestazioni tramite una fase di monitoraggio

Le prestazioni di un'applicazione cloud sono determinata dall'interazione di diversi fattori in nel ambiente cloud. A causa della complessa architettura del ambiente cloud, è difficile capire come questi fattori interagiscano tra di loro e in quale misura contribuiscono alle prestazioni di una applicazione.
Il modello [20] tenta di modellare tutti i fattori che influiscono sulle prestazioni. Il modello introdotto in questa sezione è schematizzato in figura 2.1.

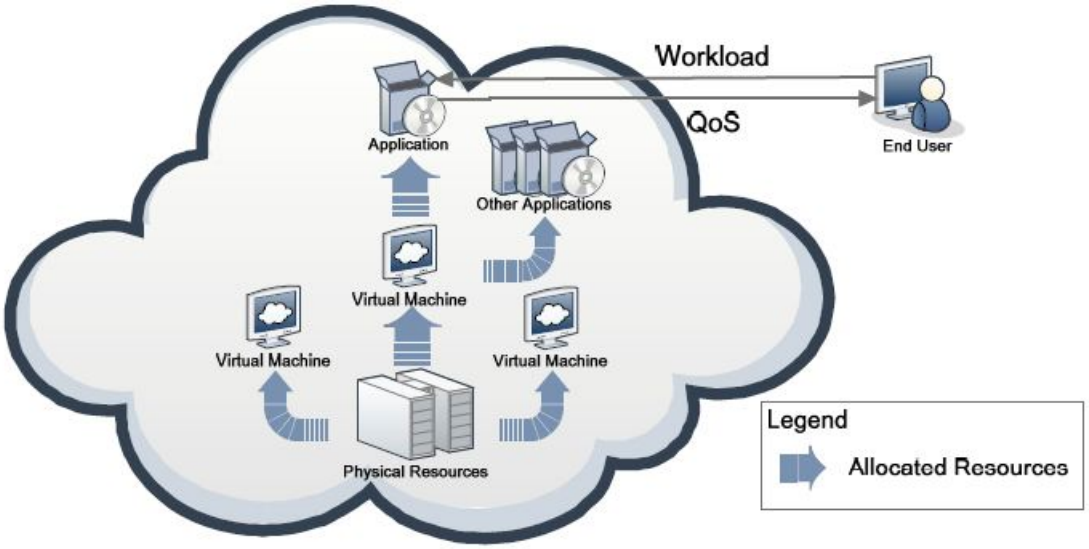

Figura 2.1: Modello per garantire le prestazioni di applicazioni Cloud.

Questo modello parte da due attributi di QoS, l'avviabilià e dal tempo di risposta[1], per determinare un valore quantitativo di performance. Per questi due parametri si definisce una metrica.
L'avviabilità si calcola come rapporto tra uptime e la somma di uptime con downtime. Invece il response time si presenta come la devianza standard dal tempo medio di risposta, questo valore da un'idea sulle varianzioni delle prestazioni. Basandosi su questi due paramatri si ottiene una metrica per le prestazioni. Infatti le prestazioni vengono calcolate come rapporto tra avviabilità e tempi di risposta.
Questo modello funziona come segue. Di solito, c'è un certo numero di macchine virtuali in una rete cloud. Un'applicazione sul cloud è distribuita su un server di

[1]Tempo di risposta è il tempo trascorso dal momento nel quale l'utente invia una richiesta e a quando la richiesta viene soddisfatta.

applicazioni in una macchina virtuale. A causa della limitazione delle risorse, altre applicazioni possono essere distribuite nella stessa macchina virtuale. Il risultato di questo modello rappresenta la performance di una applicazione cloud specifica, che si base anche su un aglomerato di altri attributi.

Questi attributi comprendono anche le risorse assegnata alla macchina virtuale in cui risiede l'applicazione, le altre applicazioni che si trovano sulla stessa macchina virtuale, le risorse effettivamente utilizzate dall'applicazione, il carico di lavoro e così via. Il valore di questi attributi e le prestazioni del sistema possono essere raccolti da un monitoraggio continuo. Tutti questi parametri vengono confrontati con i rispettivi valori nel SLA, e si determinano gli attributi da sistemare per soddisafre la richiesta di prestazioni.

La modellazione delle prestazione è un processo iterativo. Primo, si ottengono i dati di un training set monitorando continuamente l'esecuzione su cloud. La performance è valutata in base alla sua metrica e l'eventuale configurazione (ad esempio, la quantità di risorse allocate). I dati nel training set sono file con valori predeterminati. Poi un paio di algoritmi di apprendimento automatico sono applicati su questi dati, per trovare un modello di prestazioni che riesce a capire il rapporto tra questi attributi e le prestazioni dell'applicazione cloud, di seguito si ottiene un modello applicabile. Sulla base di questo modello, le prestazioni di un'applicazione cloud possono essere calcolate. Questo modello può anche stabilire come modificare la configurazione e l'allocazione delle risorse nell'ambito di un requisito di prestazione fornita.

Allora il primo passo sarà ottenere i dati del training set. Questi dati sono raccolti con il monitoraggio continuo e si fa ad interavlli di 5 minuti sui seguenti attributi: memoria totale, numero processori utilizzati (sia virtuali che fisiche), numero di applicazioni su macchina virtuale, utilizzo CPU, consumo risorse, numero di connessioni per minuto.

Applicando un algoritmo di regressione lineare per determinare un valore delle prestazioni. Questo risultato si può usare per determinare lo stato corrente delle applicazioni e consecutivamente gli attributi possono essere sistemati in maniera adattiva per rispondere alla richiesta per le prestazioni.

Infatti questo approccio più essere efficace ma si deve tenere conto del fatto che

tutti i requisiti di SLA devono essere monitorati continuamente. Con l'aumentare del numero di servizi resi, questo fatto introduce un fattore che può influenzare la performance, e richiede al provider del servizio di aggiungere ulteriori infrastrutture per garantire che la fase di monitoraggio non abbia influenza su quello che l'utente percepisce.

2.2 Prestazioni su CERAS

La figura 2.2 è una schematizzazione del laboratorio cloud del Centre of Excellence for Research in Adaptive Systems (CERAS) [22]. Lo sviluppatore dell'applicazioni sistema il codice sorgente continuamente, scoprendo e migliorando le operazioni inefficienti. Il provider del cloud modifica la ripartizione delle risorse di una determinata applicazione per mantenere la sua QoS ed offrire un costo inferiore. Lo scopo di un modello di prestazione è di predire l'effetto delle modifiche a molte

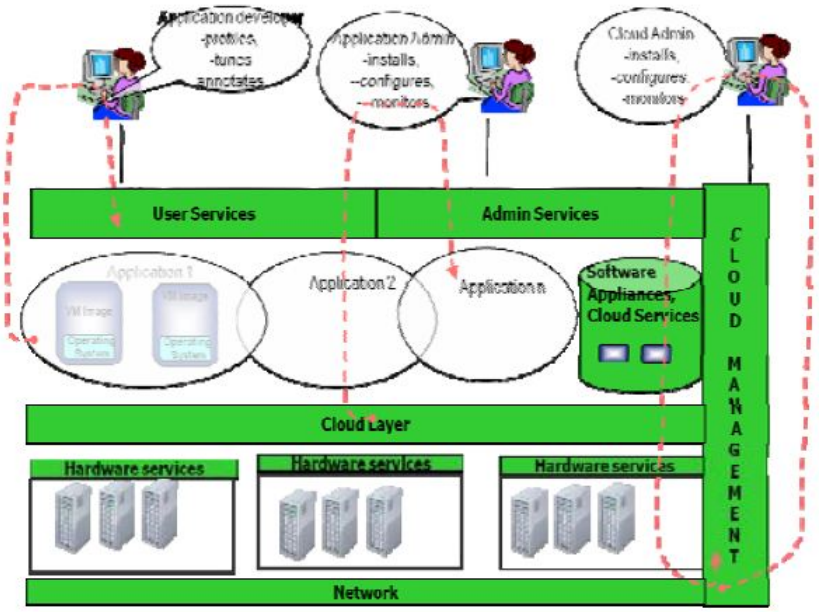

Figura 2.2: Cloud CERAS.

variabili decisionali, e per aiutare a prendere decisioni ottimali su di esse. In figura

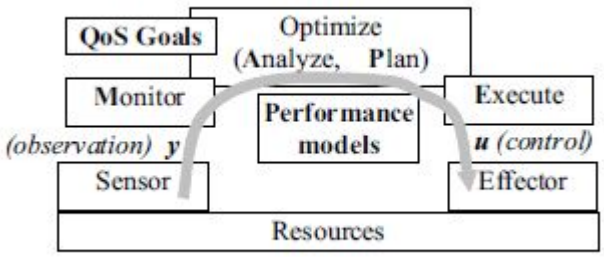

Figura 2.3: Controllo del feedback per il QoS e l'ottimizzazione.

2.2 sono mostrati tre cicli di feedback per CERAS. Ognuno di questi cicli può avere una struttura come quella mostrata in figura 2.3. I camabiamenti occorrono in base alle variabili che devono essere controllate e al modello usato per le decisioni. Lo sviluppatore modifica il codice utilizzando modelli intuitivi per l'ottimizzazione delle prestazioni. I modelli utilizzati dall'amministratore possono essere semplici feedback dei cicli. Di per sè questo livello non può avere la capacità sufficiente a garantire QoS. Le attività di ottimizzazione eseguiti dagli amministratori delle applicazioni e del cloud possono essere attuate attraverso gestori autonomi. In questo modello l"amministratore del cloud utilizza una Layered Queueing Networks (LQN) [23] e prende decisioni sulle le risorse da assegnare a ciascuna applicazione.
Il modello di prestazione si completa con dati ottenuti in fase di runtime, forniti dai sensori. Il modello supporta la modificazione delle applicazioni in ogni momento ed è relativamente veloce e poco costoso, quindi è il modo migliore per raggiungere gli obiettivi di QoS. Tuttavia può non essere in grado di per sé di mantenere un SLA, e non può fissare i propri obiettivi di QoS.

Il Cloud incorpora diversi elementi per fornire il controllo di feedback mostrato in figura 2.3, e l'architettura risultante è rappresentato nella figura 2.4. Il monitoraggio delle risorse viene effettuato e da dati sul livello di utilizzo di processore fisico, macchina virtuale, e altre risorse logiche. Il monitoraggio delle richieste effetuate dagli utenti da dati riguardanti velocità e tempi di risposta. Il medello di pretazioni memorizza un sotto-modello per ogni tipo di applicazione su come l'applicazione viene distribuia, inoltre include strumenti di misurazione per aggiornare i parametri del modello in maniera periodica partendo dalla fase di monitoraggio(il ModelTracker). Quando una nuova applicazione viene caricato un modello iniziale di prestazioni viene fornito dal programmatore dell'applicazione, derivata dalla progettazione di applicazioni o tracciando il suo comportamento. Infine alcuni strumenti di distribuzione devono essere inclusi per caricare e inizializzare l'immagini di macchine virtuali sui processori di hostaggio.
Questo modello sintetizzata un approccio di ottimizzazione.
Si consideri una nuvola con mille macchine virtuali con un carico che va da una a

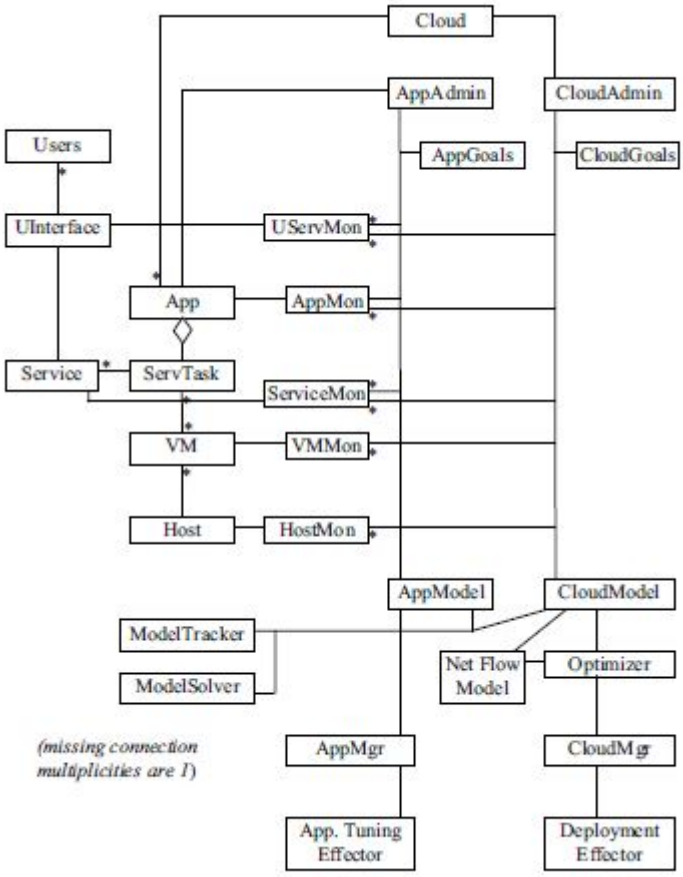

Figura 2.4: Architettura per l'ottimizzazione.

cinquanta applicazioni. Ogni applicazione ha la stessa architettura e il LQN mostrato in figura 2.5, ma diversi parametri e requisiti. La distribuzione incrementale pone ogni applicazione in un numero sufficiente di processori per soddisfare i suoi vincoli di QoS, in modo disgiunto dal numero di processori già assegnati. La completa installazione consente di ottimizzare e determinare anche un eventuale profitto partendo da una metrica presentata dal modello.

Lo svantaggio dell'ottimizzazione completa è il lavoro in più effettuato per ridistribuire le applicazioni esistenti ad ogni passo.

Tuttavia il suo vantaggio è che può massimizzare il profitto e garantire fino ad un certo punto le prestazioni stando ai requisiti di QoS presentati. In questo modello si può decidere cosa si è interessati ad espandere, in caso di profitto si da precedenza alle aplicazioni più reditizie, oppure ridistribuendo potenza di calcolo alle aplicazioni.

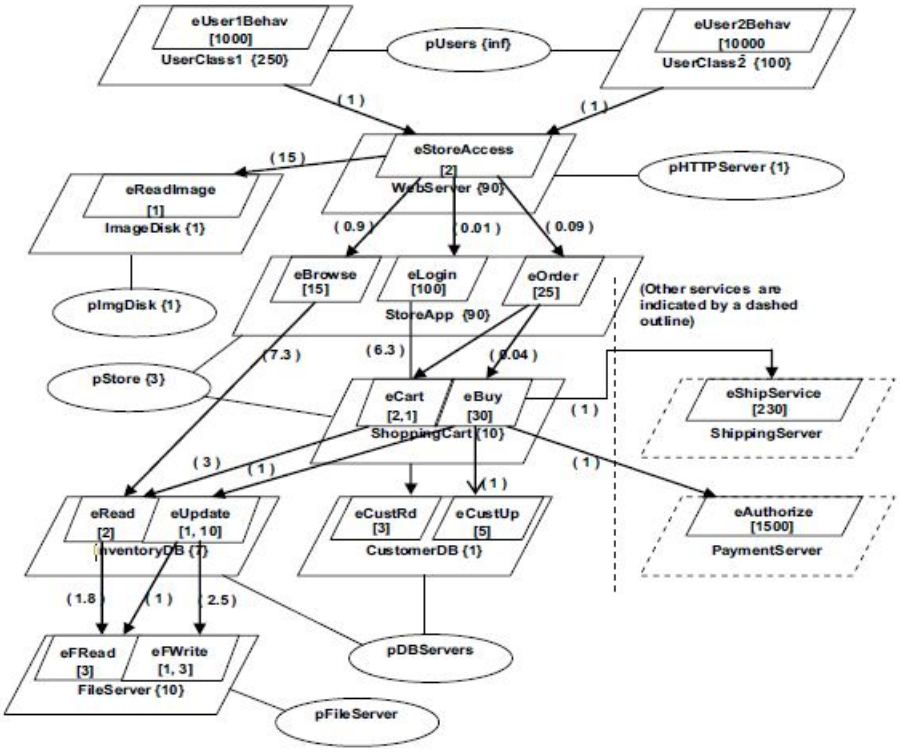

Figura 2.5: Applicazione web-based da lanciare sul cloud.

Altre politiche possono essere attuate, coinvolgendo per esempio vincoli aggiuntivi per classe.

2.3 Approvigionamento delle macchine virtuali

L'pprovigionamento su cloud [24] è il processo di implementazione e gestione di applicazioni su infrastrutture cloud. Si compone di tre fasi principali

- *Approvigionamento delle macchine virtuali.* Comporta la creazione di istanze di una o più macchine virtuali (VM) che corrispondono alle caratteristiche hardware specificate e soddisfano i requisiti del software di un'applicazione. La maggior parte dei fornitori di servizi cloud offrono una serie di generiche classi di VM con il software generico e le configurazioni delle risorse. Per esempio Amazon EC2 supporta 11 tipi di macchine virtuali [5], ognuna con diverse configurazioni di processori, memoria, e prestazioni in I / O.

- *Approvigionamento di risorse.* È la mappatura e la programmazione della macchine virtuale nei server cloud. Attualmente, la maggior parte dei provider IaaS non forniscono alcun ai fornitori di applicazioni sul controllo del approvigionamnto. In altre parole, la mappatura delle macchine virtuali su server fisici è completamente nascosta ai fornitori di applicazioni.

- *Approvigionamento delle applicazioni.* È lo sviluppo di applicazioni specifiche (come il sistema ERP, BLAST e web server) all'interno di macchine virtuali e la mappatura delle richieste degli utenti finali alle istanze delle applicazioni.

Il modello presentato [25] in questa sezione si concentra sul approvigionamento delle macchine virtuali e delle applicazioni. L'approvigionamento di applicazioni su VM sono processi che i fornitori di servizi applicativi sono grado di controllare. L'obiettivo dell'approvigionamento delle applicazione è quello di assicurare un utilizzo efficiente delle risorse IT virtuali. Tale efficenza può essere raggiunta attraverso l'utilizzo di tecniche come il bilanciamento del carico e la mappatura efficiente delle richieste dall'utente finale. Mentre l'obiettivo dell'approvigionamento di VM è quello di fornire alle applicazioni sufficiente potenza di calcolo, memoria , e prestazioni in I/O per soddisfare il livello di QoS atteso dagli utenti finali. Quest'ultimo è ottenuto sia aumentando, diminuendo la capacità delle VM distribuite o aumentando o diminuendo il numero di applicazione e di istanze di VM.

Questo approccio parte dal presupposto che le applicazioni sono ospitate all'interno di macchine virtuali per consentire la condivisione delle risorse su un'infrastruttura cloud. È possibile che un applicazione multi-tier venga eseguita su più macchine virtuali che si estendono su server di calcolo trasversali. Tuttavia, il modello presentato in questa sessione si assume che vi è un mapping uno-ad-uno tra un istanza dell'applicazione e un'istanza di VM.
Il cloud contiene data center come reti virtualizzate di IaaS (server di calcolo, database e reti) e PaaS (load balancer e auto scaler), in modo che gli operatori siano in grado di accedere e distribuire, su richiesta, applicazioni (SaaS) da qualsiasi parte del mondo, a costi competitivi e con requisiti di QoS garantiti. Il sistema di cloud computing [26], P, è un insieme di infrastrutture cloud di proprietà e gestito da IaaS/PaaS terziari come Amazon EC2, Flexiscale e GoGrid. Più formalmente, $P = (c_1, c_2, \ldots, c_n)$, dove $c_1, c_2, \ldots, c_n$ sono i data center che fanno parte di P.
La distribuzione di un applicazione si compone di m istanze di VM, $v_1, v_2, \ldots, v_n$, dove m è fissa o varia nel tempo in base al carico di lavoro corrente e i requisiti di prestazioni. Le istanze di applicazioni sono esempi di software SaaS che possono essere di proprietà di piccole o medie imprese (PMI) o governi che usano le proprie applicazioni tramite infrastrutture cloud. Qua si considera solo il caso in cui le applicazioni(SaaS) e le piattaforme(PaaS) sono parte in una organizzazione di cloud, invece le risorse si trovano in un'altra organizzazione .

Si considera s_j un elemento di un applicazione dell'utente finale che esegue un azione o funzione specifica . Le azioni o funzionalità variano in base al modello dell'applicazione. Ad esempio, un public computing come Folding@home e SETI@home forniscono funzionalità per l'esecuzione di modelli matematici in un dato insieme di dati, mentre un web server è un servizio che fornisce contenuti, ad esempio pagine web, utilizzando il Hypertext Transfer Protocol (HTTP) su Internet. Eventuali dipendenze tra le richieste si presumono gestite dall'utente. Pertanto, dal punto di vista di s_j, le richieste per azioni o funzionalità sono indipendenti l'una dall'altra. Questo è il caso, per esempio, dell'elaborazione di richieste HTTP. Anche se alcune informazioni sullo stato delle sessioni possono essere memorizzati nel Cloud,

le dipendenze (ad esempio, eccezioni di sicurezza e protocolli di comunicazione) sono gestiti dal lato utente tramite browser web.
I fondamentali carichi di lavoro in scienza, fisica, commercio, media, e ingegneria possono essere modellati come carichi di lavoro *embarrassingly parallel o Bag of Tasks (BoT)*, i task dei quali sono indipendenti. Alcuni esempi popolari in questo campo sono le applicazioni scientifiche come BLAST, MCell e INS2D, ed anche applicazioni di public computing come Einstein@home, Folding@home, e SETI@home. Più specificamente si tratta di applicazioni composte da task indipendenti che possono essere modellate come richieste di servizi, e inviate dagli utenti finali ad una istanza dell'applicazione.
I web server e l'esecutore dei carichi di lavoro BOT sono esempi di altri tipi di modelli d'applicazione. In entrambi i casi, un carico di lavoro G_s è composto da h richieste indipendenti $\{r_1, r_2, \ldots, r_h\}$(vale a dire, non c'è comunicazione tra le richieste e il calcolo richiesto da ciascuna richiesta non è distribuito) che vengono ricevuti da istanze di applicazioni in istanti $\{t_1, t_2, \ldots, t_h\}$. Inoltre, tutti i requisiti del software e dei dati per l'esecuzione di una richiesta è a carico della VM in esecuzione, questo significa che l'esecuzione delle richieste di servizio non richiede l'utilizzo di cloud storage.
In questo modello i requisiti di QoS relativi ad un'applicazione includono il tempo di risposta T_s e il tasso di rigetto $Rej(G_s)$ delle richieste. Questi parametri sono importanti perché hanno effetto diretto sulla percezione delle prestazioni sull'applicazione SaaS sa parte dell'utente. Se il tempo di risposta è troppo elevata o se le richieste vengono respinte, alcuni utenti potrebbero abbandonare l'applicazione in modo permanente, questo diffetto è fonte di pubblicità negativa per il provider può causare la perdita di una parte del flusso di entrate (costo di utilizzo e la pubblicità). Pertanto, soddisfare la QoS per un'applicazione è fondamentale.

L'architettura mostrata nella figura 2.6 affronta problemi di comportamento incerto, errori di stima, e del carico di lavoro dinamicamente distribuito relativo all'approvigionamento su Cloud. Componenti software dell'architettura sono gestiti dal fornitore del servizio. Il suo strato SaaS contiene un meccanismo di controllo

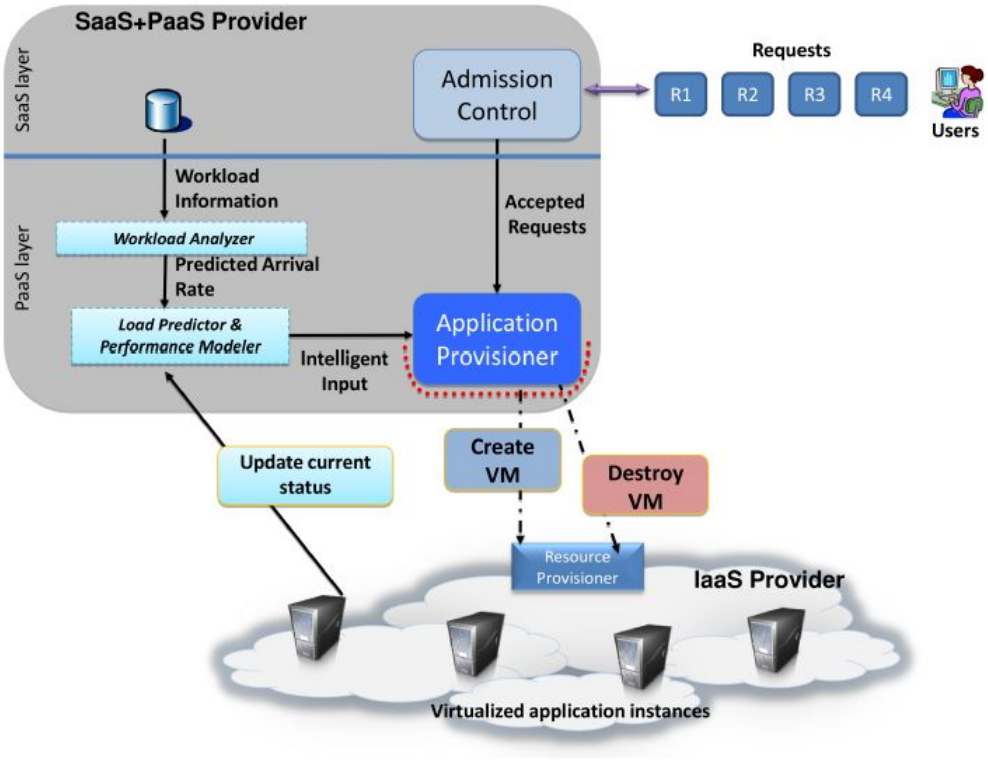

Figura 2.6: Meccanismo adattivo per l'approvigionamento delle macchine virtuali.

per accettare le richeste in base al numero istanze per applicazione. Ovvero, se tutte le istanze di applicazioni virtaulizzate hanno k richieste nelle loro code, nuove richieste vengono respinte, in quanto possono comportare una violazione T_s. Le richieste accettate vengono inoltrate allo strato PaaS del provider, che implementa il sistema proposto. Principalmente, i seguenti componenti sono fondamentali per la funzionalità complessiva del sistema:

1. *Approvigionatore di applicazione*, punto di contatto nel sistema che riceve le richieste accettate e occupa macchine virtuali e istanze delle applicazioni in base ai risultati ottenuti dal "analisi del carico di lavoro" e dal "anticipazione del carico e modello per le prestazioni". Questa componente, crea nuove istanze in base ai requisiti di QoS.

2. *Analisi del carico di lavoro*, genera la stima delle future richieste per l'applicazione. Le informazioni ottenute vengono utilizzate per calcolare il numero esatto istanze di applicazione necessari per il conseguimento degli

obiettivi di QoS. L'anticipazione può essere basata su informazioni diverse, ad esempio, può basarsi su dati storici su utilizzo delle risorse, o sulla base di modelli statistici derivati dalle distribuzioni note di carichi riguardanti le applicazioni. Queste informazioni vengono passate al sistema di "anticipazione del carico e modello per le prestazioni".

3. *Anticipazione del carico e modello per le prestazioni.* che risolve un modello analitico basato sulle prestazioni del sistema osservato e predetto carico di decidere il numero di istanze di macchine virtuali che dovrebbe essere assegnata ad un applicazione. Questa unità è responsabile di decidere il numero di istanze di applicazioni virtualizzate necessarie per soddisfare gli obiettivi di QoS.

Questo meccanismo è sempre in funzione per garantire che gli obiettivi per l'approvigionamento siano rispettati in ogni momento. L'approvigionamento viene effettuato in base ai seguenti obbiettivi:

- *Automazione:* tutte le decisioni relative alla fornitura devono essere effettuati automaticamente, senza intervento umano;
- *Adattamento:* L'approvigionamento delle applicazione deve adattarsi alle incertezze, come cambiamenti dell'intensità del carico di lavoro;
- *Garanzia di prestazioni:* L'allocazione delle risorse nel sistema può variara dinamicamente per assicurare il conseguimento degli obiettivi di QoS.

Questo meccanismo si avvale delle prestazioni analitiche (sistema basato su un modello di accodamento) e le informazioni del carico di lavoro per prendere decisioni per effettuare approvigionamenti per le applicazione. L'approccio proposto è in grado di modellare l'infrastruttura utilizzando solo le informazioni fornite dall'IaaS che i fornitori del servizio mettono a disposizione per i clienti e dei dati di monitoraggio delle VM in esecuzione. L'obiettivo del modello è quello di raggiungere i requisiti di QoS in termini di prestazioni, tempi di servizio e tasso di rifiuto delle richieste, e utilizzo delle risorse disponibili.
Risultati sperimentali ottenuti da simulazioni sul carico di lavoro indicato che questa tecnica di approvigionamento è in grado di rilevare variazioni di intensità del carico

di lavoro (modello d'arrivo e di richieste di risorse) che si verificano nel corso del tempo e riesce allocare le risorse di conseguenza per raggiungere gli obiettivi di QoS e prestazioni delle applicazioni.

Capitolo 3

Scalabilità

La scalabilità è fondamentale per il successo di molte organizzazioni di servizi web attualmente sul mercato e nei sitemi che contengono informazioni che possono, improvvisamente, diventare molto richieste.
La scalabilità è definita in modi diversi in varie letterature. Secondo [13] una definizione generalizzata è la seguente:

La scalabilità di un servizio è una proprietà che fornisce una stima della capacità di gestire crescenti quantità di carichi di servizio senza subire degrado della qualità. Inoltre, la scalabilità ottenuta come resultato dell'applicazione di modelli o schemi (come modelli che adeguano le risorse in modo appropriato) deve essere proporzionale al costo per applicare il modello.

In questa definizione, spuntano quattro elementi:

- gestione dei crescenti carichi di lavoro,
- garantire la scalabilità attraverso modelli,
- costi accettabili,
- evitare la degradazione significativa della QoS .

I gruppi di consumatori di servizi generalmente non sono noti in anticipo. Cioè, non è possibile prevedere in maniera statistica il numero di consumatori di servizio

e la quantità di invocazioni dei servizi. In presenza di gandi crescite dei carichi di un servizio, la qualità dei servizi ne subira immediatamente il peso. Quindi, la scalabilità deve risolvere problemi di gestioni di cambiamenti improvvisi dei carichi sui servizio.
Garantire la scalabilità in presenza di carichi elevati di servizio ha un elevato costo. Essa può essere assicurata mediante l'addozione di modelli per garantire la scalabilità. Lo schema più comune è quello che aggiunge risorse con il variare della domanda.
C'è sempre un certo costo nella gestione di sistemi per garantire la scalabilità (per esempio: sistemi che aggiungono ulteriori CPU e memoria, server, rete, etc.). La scalabilità acquisita attraverso tali sistemi deve essere proporzionale al costo per applicare il sistema.
I servizi devono fornire un certo livello di qualità, come specificato nel loro SLA. Quindi, servizi con buona scalabilità non dovrebbero avere problemi nel rispettare i loro minimi requisiti di QoS.

In questa sezione verranno introdotti e commentati alcuni modelli di rilevanza che trattano alcuni aspetti fondamentali per garantire la scalabilità come elemento o supporto alla QoS.

3.1 Un approccio software alla scalabilità

I sistemi di servizio sono costituiti da più nodi che vengono distribuiti sulla rete. Pertanto, i consumatori di servizi non conoscono l'ubicazione del nodo che utilizzano. Per costruire sistemi con servizi scalabil, sono indispensabili componenti di gestione in grado di monitorare lo stato corrente dei nodi. Questo modello utilizza [13] due componenti per gestire la scalabilità dei servizi, il Global Scalability Manager (GSM) e Global Scalability Manager (RSM). Sulla base di queste due componenti si costruisce il sistema scalabile mostrato in figura 3.1.

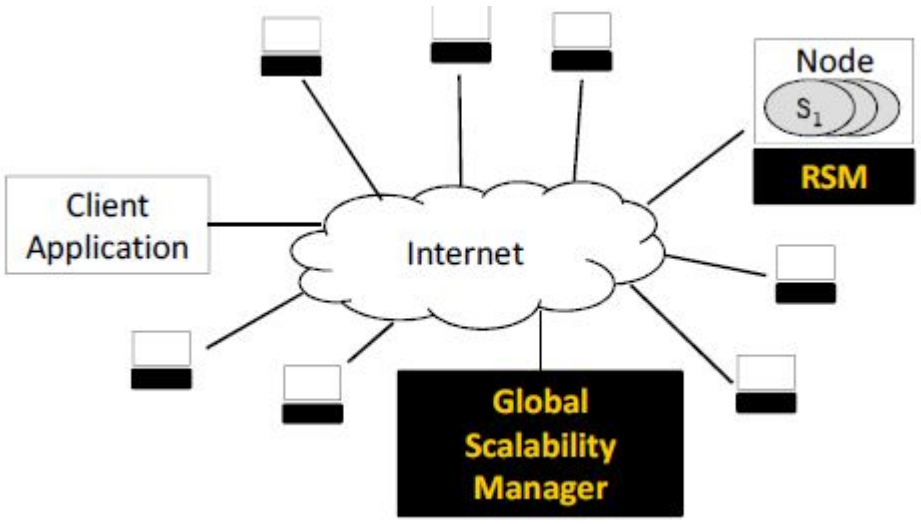

Figura 3.1: Sistema scalabilie di servizi.

GSM è una componente fondamentale di questo modello per gestire la scalabilità dei servizi. GSM effettua operazioni di bilanciamento del carico sul sistema. A tal fine, GSM acquisisce lo stato attuale dei nodi che sono registrati nel sistema di servizi,e pianifica un metodo per garantire la scalabilità.
RSM è un componente di gestione installata su un nodo. Un RSM osserva lo stato del nodo e lo consegna al GSM, inoltre esegue anche le operazioni per garantire la scalabilità in base alle istruzioni del GSM.

Il modello discusso in questa sessione garantisce la scalabilità in modo dinamico e autonomo. Il processo per garantire la scalabilità è mostrato in figura 3.2.

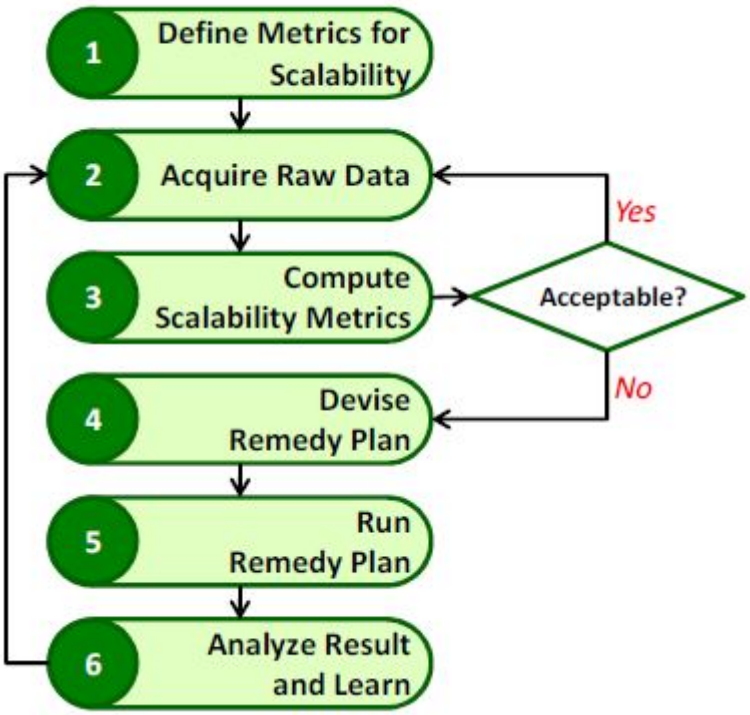

Figura 3.2: Applicazione web-based da lanciare sul cloud.

- *Il primo passo* è quello di definire le metriche di qualità per la misurazione della scalabilità dei servizi. Ad esempio, il throughput è un parametro che misura l'efficienza nel gestire le invocazioni di servizio in un dato intervallo di tempo. Metriche definite in questa fase vengono utilizzati in due modi: primo come base per scegliere dati che devono essere raccolti dai servizi, e secondo per garantire la scalabilità nel terzo passo.

- *Secondo passo* è acquisire l'insieme di dati grezzi ottenuti dai servizi monitorati.

- *Terzo passo* è quello di calcolare le metriche di scalabilità. Se i risultati ottenuti danno un livello accettabile di scalabilità, si ritorna al secondo passo. Se le metriche risultanti indicano la necessità di intraprendere azioni per sistemare la scalabilità, si esegue il quarto, quinto e sesto passo.

- *Quarto passo* è quello di elaborare un piano per migliorare subito la scalabilità. Il piano si basa sugli stati attuali dei servizi monitorati. Il piano decide coa si deve aggiungere per garantire le richieste del servizio.

- *Quinto passo* è quello di ottenera la scalabilità desiderata mediante il piano fissato nel passo precedente.

- *Sesto passo* è quello di analizzare il risultato ottenuto applicando il piano. Se il risultato è quello desiderato, il piano e le circostanze di sistema che hanno compromesso la scalabilità vengono salvate per per essere utilizzate in futuro, aggiungendo un fattore di intelligenza al modello.

3.2 Scalabilità per applicazioni web

In questa sezione si introduce un metodo [14] per generare una politica valida per la scalabilità delle applicazioni multi-tiers su cloud. Questo approccio si basa su un modello accurato di prestazioni e un metodo euristico di ricerca.

Applicazioni sul cloud non sono mai presentate e costruite nella stessa maniera, hanno diversi requisiti di QoS e carichi di lavoro dinamicamente variabili. In questo modella la QoS è descritta come SLA di ogni classe di operazioni.

Il modello generalizzato è presentato in figura 3.3 dove si può vedere anche la componente principale del modello, *cloud application scalability controller.* Il cloud

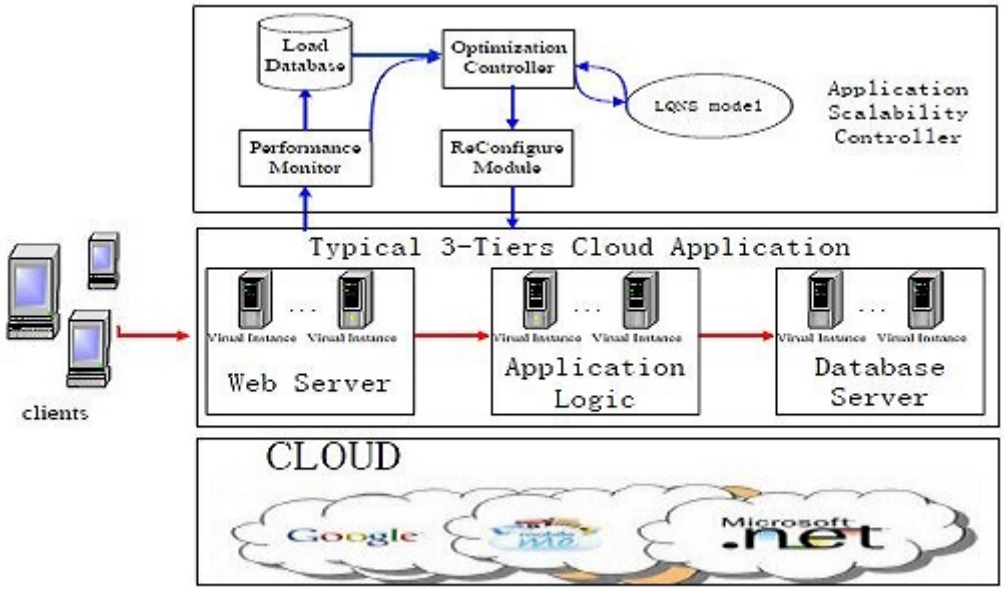

Figura 3.3: Sistema scalabilie di servizi.

application scalability controller è composto da un monitor delle prestazioni, un database per il carico di lavoro, LQN model solver [23], modulo di riconfigurazione e modulo di ottimizzazione.

Il monitor delle prestazioni raccoglie i dati sui parametri del carico di lavoro e le prestazioni (ad esempio, il tempo di risposta e la velocità) di ogni transazione e li memorizza nel database per il carico di lavoro. Il modulo di riconfigurazione deve, principalmente, ridistribuire le applicazioni sul cloud, aggiungendo o riducendo server

virtuali a secondo dello schema di configurazione ottimizzato.

LQN model solver crea modelli per le prestazioni delle applicazioni cloud multi-tires, e fa previsioni precise circa le loro prestazioni su qualsiasi carico di lavoro o configurazione.

Nel modulo di ottimizzazione i parametri di prestazioni sono combinati con i requisiti di SLA per calcolare la funzione di utilità. Il modulo di ottimizzazione implementa un metodo di ricerca euristico, che combina un insieme di regole, il SLA e un efficiente algoritmo di pruning, e ottine uno schema ottimizzato per carico di lavoro corrente. Inoltre, il modulo di ottimizzazione viene richiamato ogni volta che le variazione del carico di lavoro superano la soglia preimpostazione.

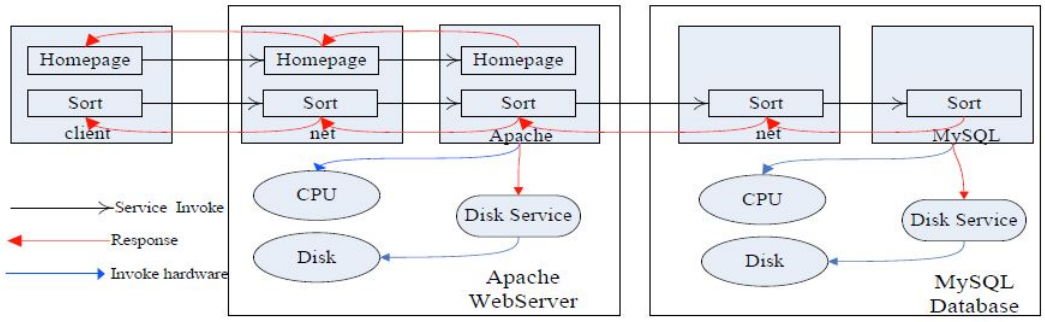

Figura 3.4: Modello LQN per un motore di ricerca di libri.

Il modulo di ottimizzazione delle applicazioni viene utilizzato per stimare i parametri di rendimento di tutte le transazioni in un sistema multi-livello sul cloud. Nel modello LQN, si ritiene che il software nel sistema multi-livelli sia composto da divers task. Ogni task ha una o più voci di servizio che rappresentano differenti pacchetti di operazioni che può eseguire. Per esempio, un dispositivo a disco può eseguire più di una operazione in tempi di servizio differente, come lettura e scrittura. Inoltre, ogni task può fare richieste che devono essere accodate in altri task.

L'essenza del modello a livelli è determinare le richiesta annidate e per ogni task si modella con una coda $M/M/n$, dove n rappresenta il numero di componenti hardware e software concurrenti e M il numero di transazioni possibili.

In figura 3.4, è illistrato il modello di prestazioni a due strati per un motore di ricerca di libri. Questo motore di ricerca è composto da 9 transazioni.

Quando aumenta il carico di lavoro e il sistema fa fatica a rispondere e diventa

indispensabile scalare le risorse in maniera efficiente. Questo modello fa questo ottimizzando la distribuzione delle applicazioni mediante un algoritmo euristico. I dati sperimentali forniti in [14] verificano la funzionalità del modello e indicano che essi riesce ad ridistribuire i carichi di lavoro per rispettare le condizioni di scalabilità imposte dal SLA.

3.3 P2P Cloud

Sia il cloud computing che il P2P sono le tecnologie Internet più diffuse negli ultimi dieci anni. Hanno vantaggi simil, come la potenza di calcolo virtualmente infinita e spazio di archiviazione, ma in realtà sono diverse.
Il cloud computing utilizza infrastrutture Client\Server (C\S) per offrire tre tipi di servizio (SaaS, PaaS, IaaS). Nel cloud i server sono in grado di offrire una grande avviabilità ad un elevato numero di utenti concorrente, in cambio di un costo(economico).
Le risorse P2P sono immense e gratuite, ma la loro disponibilità non è sempre garantita, e la tecnologia P2P è ampiamente usato nei sistemi di filesharing come Emule e BT, in streaming-video come PPlive e PPStream, e le comunicazioni di rete, come MSN e QQ.
Il modello commentato in questa sezione [15] combina queste due infrastrutture per garantire la scalabilità come requisito di QoS.
L'architettura C\S è di gran lunga la struttura più comune per il cloud computing. In C\S, tutti i server sono affidabili, tutti i dati sono memorizzati in banche dati centrali e sono accessibili. Inoltre i server sono in grado di sostenere la frequente lettura e scrittura da un grande numero di utenti. Un cluster di potenti server può facilmente gestire un elevato numero di task concorrenti. Ma, quando il numero di utenti simultanei continua a crescere, i server centrali devono fare fronte ad un enorme traffico e si può faccilmente incorrere in errori. Tali difetti mostra il limite della scalabilità e flessibilità delle infrastrutture C\S. In conseguenza nuovi server per il cluste devono essere comprati per fare fronte alle richieste.
Inoltre, il cloud computing devono fare affidamento su connessioni ad alta velocità e a banda larga in modo da communicare con gli utenti finali. Una volta che i server cloud o la rete non si connettono, gli utenti finali non ottengono alcun servizio.
L'architettura P2P è intrinsecamente scalabile e può scaricare il lavoro dei i server centrali sfruttando la potenza di calcolo degli utenti.

P2Pcloud [15] rappresenta un cloud di storage che integra le tecniche P2P in cloud computing. In P2Pcloud, oltre ad un nucleo cloud che garantisce un servizio affidabile, vi è un nuovo cloud estensibile, che si compone di nodi "eccellenti" e nodi

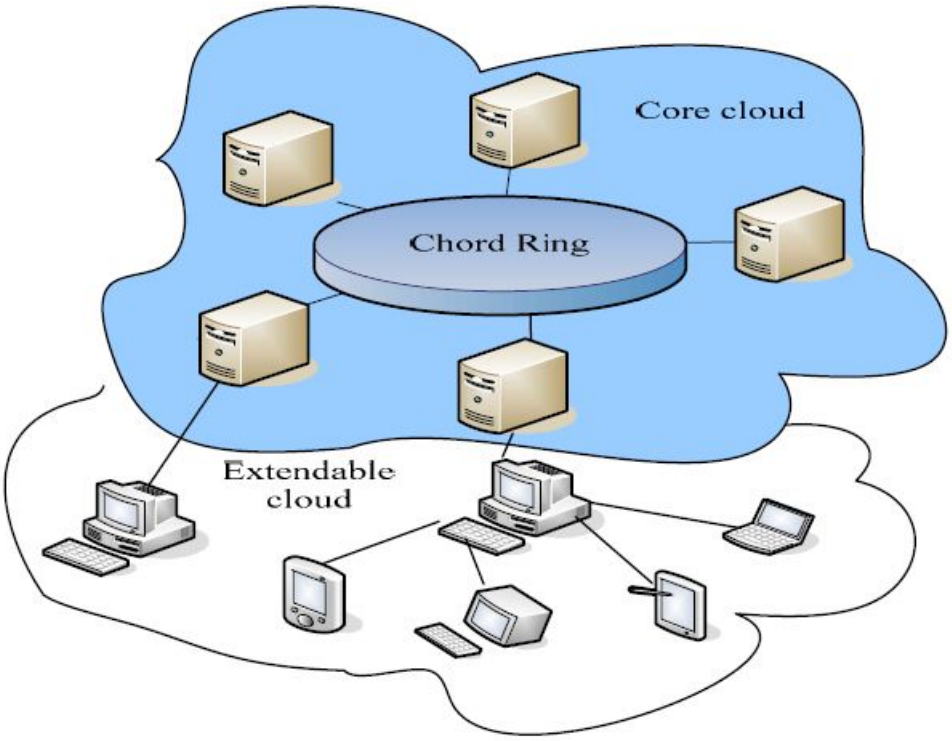

Figura 3.5: Struttura del P2Pcloud.

ordinari. Grazie a basso costo di gestione delle risorse nel cloud estensibile, il carico sul nucleo cloud si allieva, in tal modo il nucleo cloud è in grado di fornire servizi di memoria e supporto più affidabili. Integrando il cloud e il P2P, P2Pcloud trasferisce efficacemente carico dal nucleo cloud al cloud estensibile, ma in modo non percepibile aggiunce carico ai nodi ordinari, quindi minor costo di costruzione e di manutenzione del cloud, e si ottiene più scalabilità.

Il nucleo cloud si basa su un anello, fornisce servizi affidabili di supporto, ad esempio un indice globale di risorsa o repliche di memoria, e il cloud estensibile composto da un numero immenso di nodi ordinari che offron, principalmente, servizio di base, come lettura, scrittura dati e altre esigenze frequenti degli utenti.

Come mostrato in figura 3.5, l'anello consiste principalmente di nodi con capacità elevata e buona affidabilità, mentre i nodi ordinari sono organizzanti in maniera autonoma su un albero a più livelli. L'albero si basa su una topologia P2P non strutturata e nel cloud estendibili gestisce la moggior parte delle manipolazioni e interazioni. Tale topologia può trasferire il carico dal nucleo cloud al cloud estensibile,

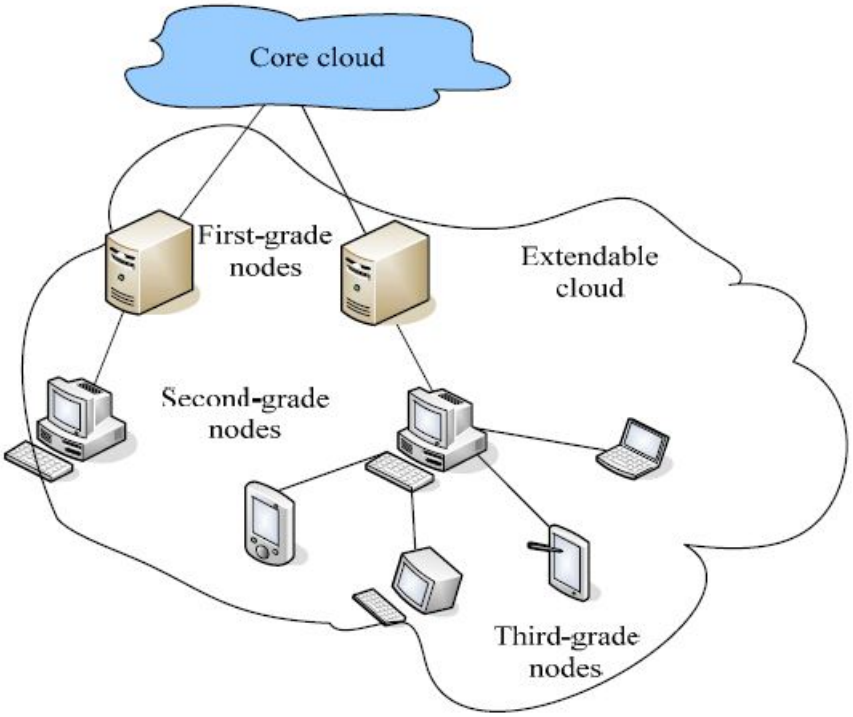

Figura 3.6: Struttura del P2Pcloud.

e distribuire il carico in maniera bilanciata e garantire un'elevata QoS per gli utenti. P2Pcloud utilizza una struttura gerarchica per gestire i nodi eterogenei. Il nucleo dipende da un anello basato su una tecnologia DHT (Distributed Hash Table) per memorizzare e recuperare gli oggetti. Con la struttura DHT, i dati possono essere assegnati e gestiti in modo efficace in tutta la rete, gli utenti possono accedere e interrogare i dati in modo efficiente con poco overhead computazionale. Il nucleo fornisce e garantisce l'usabilità, affidabilità e tolleranza ai guasti del sistema. Poiché l'anello è una struttura stabile, non necessita di frequenti interventi di manutenzione e di recupero.

La nuvola estensibile è costituito da enormi quantità di nodi comuni che spesso entrano o escono dal sistema. I nodi vengono organizzati nell'albero in base alla loro capacità (capacità di calcolo, il tempo trascorso online, dimensioni memoria e le prestazioni). I nodi con elevata capacità, sicuramente, funzioneranno meglio di un nodo a bassa capacità. Per migliorare le prestazioni di tutto il P2Pcloud, i nodi vengono classificati in diversi gradi . Se un nodo ha una capacità più elevata e

prestazioni migliori (tipo ritardo di accesso inferiore e un mantiene la connessione aperta più a lungo), il suo grado sarà maggiore di altri. I dettagli sono riportati nella figura 3.6.

I nodi sono auto-organizzati in cluster diversi. Ogni cluster è composto da nodi di grado diverso. Per garantire l'affidabilità del sistema, alcuni nodi con grado alto sono selezionati come server per comunicare informazioni tra nodi di grado inferiore e il nucleo cloud, in più comunicano metadati di supporto come informazioni di directory, indice di file e parametri fisici di nodi di basso livello e del cloud. Una volta che un nodo selezionato come server lascia la rete esso viene sostituito dal nodo di grado direttamente inferiore.

Così, come mostrato in figura 3.7, un nodo fisico sull'anello è diviso in due nodi logici. Ogni nodo logico è responsabile di alcuni nodi nel cloud estensibile. I nodi logici sono principalmente responsabili per il backup dei dati e garantiscono l'accesso ai dati e la robustezza del sistema. I guasti si possono gestire con i nodi presenti nel cloud estendibile.

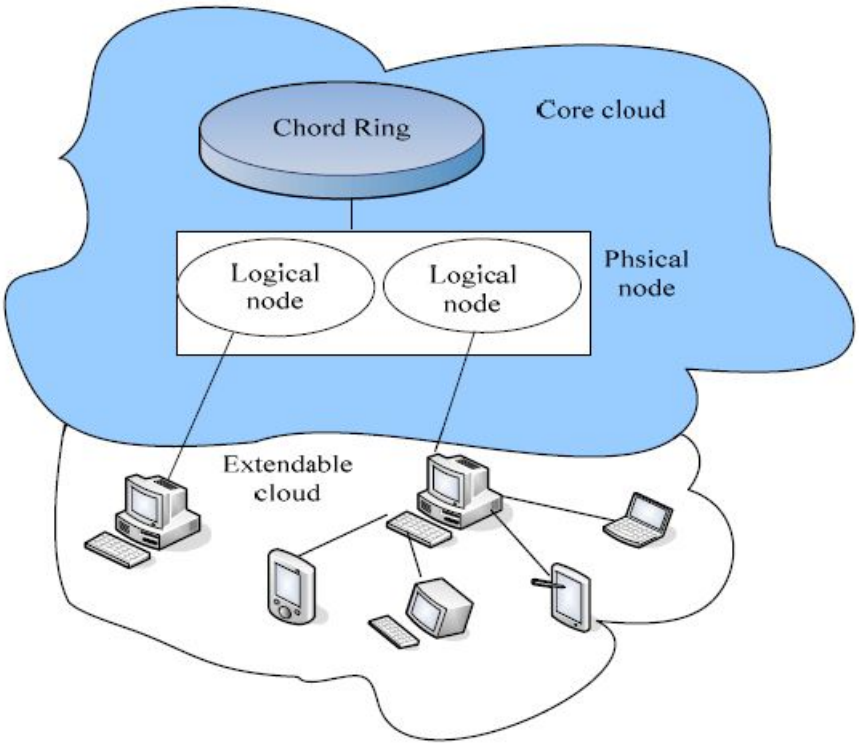

Figura 3.7: Modello LQN per un motore di ricerca di libri.

Quando i nodi dell'anello sono sovraccarichi o falliscono, l'overhead per la gestione dei dati e la latenza delle query aumenterà notevolmente. In effetti, i nodi nel cloud estensibile possono farsi carico di risorse frequenti accesse dagli utenti. Così la maggior parte del carico viene trasferito sul cloud estendibile, e l'overhead costoso per la manutenzione del nucleo viene ridotto notevolmente, inoltre il cloud etendibile aumenta la scalabilità del sistema. In base all'accesso e l'aggiornamento frequente delle risorse, i nodi della nube estensibile si organizzano in maniera autonoma in una topologia ad albero con gradi diversi come Napster, che è una struttura ibrida P2P. I nodi di alto grado possono intrinsecamente affrontare problemi hot-spot. Per esempio, quando gli utenti accedono o aggiornare alcune risorse frequentemente, il cloud estensibile gestisce tali query al posto del cloud di nucleo, quindi il bilanciamento del carico si ottiene tra nucleo e il cloud estensibile.
Rispetto alla struttura di cloud computing pura basata su C\S, P2Pcloud è più praticabile e affidabile per l'accesso di un numero elevato di utenti. Il P2Pcloud è un modello affidabile, poco costoso è soprattutto permette di garantire la scalabilità. Esso è in grado di garantire l'alta qualità dei servizi.
Unica annotazione e diffetto è la sicurezza dei dati è ulteriormente posta a rischio, questo medello aggiunge un grado di astrazione dei dati.

Capitolo 4

Disponibilità

Secondo [27] la disponibilità è il più grande ostacolo che il cloud computing affronta. Le organizzazioni si preoccupano se i servizi di Utility Computing hanno disponibilità adeguata, e su quetso il Cloud Computing lascia un pò a desiderare. Ironia della sorte, gli attuali prodotti SaaS hanno fissato uno standard elevato per la disponibilità. Oggi giorno il motore di ricerca Google è diventato quasi un sinonimo di Internet: se la gente apre Google e la ricerca non è disponibile, penserebbe subito che Internet non funziona. Gli utenti si aspettano disponibilità simile da cloud di servizi, cosa difficile da fare. La tabella 4.1 mostra le interruzioni di alcuni servizi.

Tabella 4.1: Tabella

Servizio interroto	Durata	Data
Dropbox	25/10/2012	1 ora
Gmail	22/10/2012	3 ore
Google App Engine	26/10/2012	1 ora

Anche se le aziende hanno più datacenter in diverse regioni geografiche che utilizzano fornitori di rete diversi, possono avere un'infrastruttura software e sistemi di contabilità comune per tutti i datacenter. Comunque è impossibile che un'azienda non interrompa il proprio servizio per un motivo o l'altro.
Un altro ostacolo alla disponibilità è il Distributed Denial of Service (DDoS). I criminali minacciano di tagliare i redditi dei fornitori di SaaS, rendendo i loro servizi non disponibile, estorcendo da 10.000$ a 50.000$ per impedire il lancio di un attacco

DDoS. Tali attacchi utilizzano in genere grandi botnet, prendendo bot in affitto sul mercato nero per 0,03$ a bot (simulante un utente fasullo) a settimana. Utility Computing offre ai fornitori di SaaS la possibilità di difesa contro gli attacchi DDoS utilizzando rapido scale-up. Supponiamo che un istanza EC2 può gestire 500 bot, e viene lanciato un attacco che genera un ulteriore 1GB\sec di banda finta e 500 000 bot. A 0,03$ per bot, un attacco del genere costerebbe all'attaccante 15 000$ di "investimento" in anticipo. Ai prezzi correnti di Amazon Web Services(AWS), l'attacco sarebbe costato alla vittima un extra di 360$ per ora in larghezza di banda e un extra di 100$ all'ora per la potenza di calcolo. L'attacco dovrebbe quindi durata di 32 ore, al fine di costare alla potenziale vittima più di quanto richiesto dal ricattatore. Un attacco botnet di questa durata può essere difficile da sostenere, in quanto più a lungo dura un attacco più facile da determinare e difendersi contro, e i bot attaccanti non possono essere immediatamente riutilizzati per altri attacchi sullo stesso provider. Secondo [28] la disponibilità è la probabilità che un sistema sia operativo e in grado di fornire i servizi richiesti in un dato istante. La disponibilità essere considerata e costruita con i sistemi. Essa è una fondamentale componente di QoS e deve essere offerta sempre.

In questo capitolo verranno introdotti e commentati alcuni modelli di rilevanza che trattano alcuni aspetti fondamentali per garantire la disponibilità come elemento della QoS.

4.1 Cloud@Home

Allo stato dell'arte, esistono tante infrastrutture in grado di offrire servizi IaaS, come The Eucalyptus Cloud [9] , Open-Nebula [7], etc. Il denominatore comune di queste infrastrutture, come pure di altre infrastrutture di proprietà(ad esempio, Amazon EC2, Microsoft Azure) è lo sfruttamento della potenza e l'affidabili delle infrastrutture di calcolo sottostanti.

C@H (Cloud@Home) utilizza un approccio completamente diverso [8]. C@H è un progetto italiano finanziato dal Ministero dell'Istruzione, dell'Università e della Ricerca. C@H mira a costruire un provider cloud IaaS utilizzando risorse di calcolo e di archiviazione acquisiti da collaboratori volontari. Queste risorse possono essere PC inutilizzati, piccoli cluster, generamente chiamati commercial-off the-shelf systems (COTS). Di qui, l'infrastruttura di base è costituito da risorse di calcolo disponibili in modo intermittente su reti abbastanza instabili.

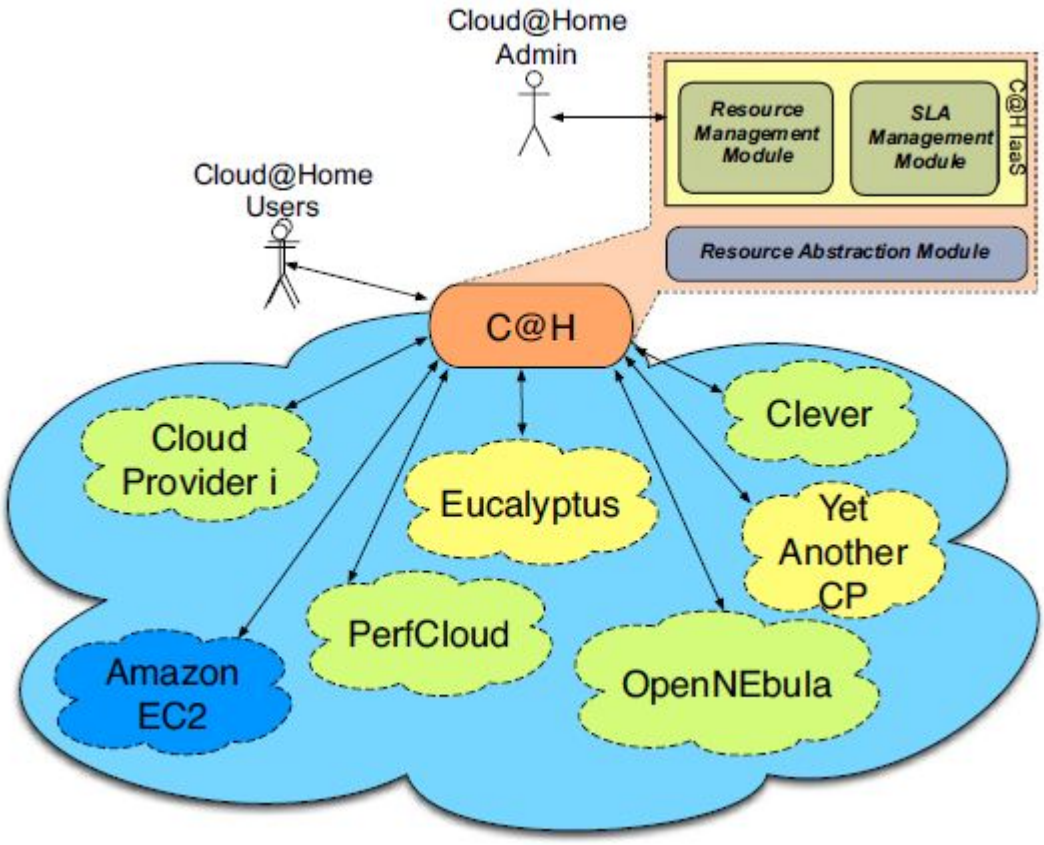

Figura 4.1: Architettura C@H.

L'insieme completo di funzionalità fornite dal provider C@H è organizzato in tre moduli, che sono raffigurati in alto a destra nella figura 4.1. Il *Resource Abstraction Module* implementa e fornisce strumenti e meccanismi di interfaccia per i IaaS Cloud Providers disposti a offrire risorse e li interconnettere il tutto. Il compito di questo modulo è quello di nascondere l'eterogeneità delle risorse raccolte dagli specifici Cloud Provider e di offrire all'utente C@H un modo uniforme per accedervi. Al *Resource Management Module* è affidata la gestione (scoperta, l'iscrizione, l'attivazione, migrazione VM, ecc) delle risorse. Il *SLA Management Module* ha il compito di negoziare con l'utente C@H il livello di qualità delle risorse richieste, poi formalizzate un SLA.

C@H fornisce ai suoi utenti strumenti e meccanismi per la gestione della QoS, soppratutto con obiettivo la disponibilità delle risorse.
L'applicazione della SLA\QoS comprende le seguenti attività:

- *negoziazione:* Gli C@H utenti possono negoziare i livelli di QoS richiesti, ricevere una garanzia formale (SLA) che il concordato livelli di QoS sarà effettivamente mantenuto.

- *monitoraggio:* I parametri non funzionali che contribuiscono al complessivo QoS sono costantemente monitorati per garantire che nessuno dei termini SLA venga violato.

- *recupero:* Azioni di recupero vengono attivate ogni volta che il QoS sta per essere violato e eventuali contromisure da adottare al fine di riportare l'QoS allo stato normale vengono intraprese.

- *terminazione:* Semplicemente si sbloccano le risorse che non sono piùutilizzate dall'utente.

Il processo inizia con la negoziazione della QoS tra le parti (utente e provider C@H). In particolare, l'utente C@H presenta una proposta di SLA al provider. Se il provider rifiuta la proposta, il processo termina. In caso contrario, la SLA è formalmente sottoscritto dalle parti. I vincoli di QoS sono estratti dal documento

SLA e sono costantemente monitorate. Se una violazione di SLA è rilevata, l'attività di ripristino si attiva immediatamente. Se il recupero non è possibile, la fornitura di servizi termina, altrimenti la QoS viene reimpostata al livello iniziale e controllata per eventuali nuovi difetti.

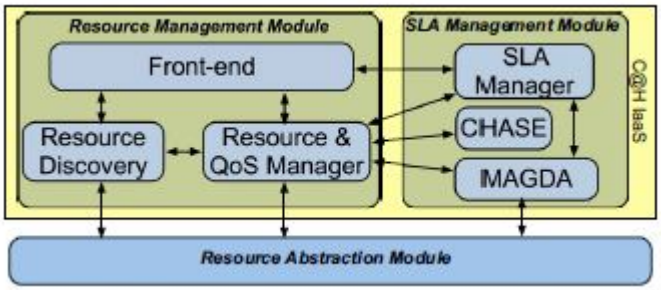

Figura 4.2: Architettura C@H per la gestione della QoS\SLA.

Da un punto di vista architettonico, il C@H modulo si decompone come in figura 4.2. L'architettura presentata si ispira al modello SLA@SOI [29]. Più dettagliate, il SLA Management Module è costituito da tre componenti: il SLA Manager, il Cloud@Home Autonomic Service Engine (CHASE) [30] e il Mobile Agent Based Grid Architecture (MAGDA) [31]. Il SLA Manager è la componente principale, cioè, il modulo che gestisce l'intero processo di gestione QoS, supportato dal Resource and QoS Manager. Entrambe le unità hanno il compito di supportare la fase di negoziazione, di coordinare il monitoraggio e le fasi di recupero.

Il modulo Resource and QoS Manager a sua volta è composto da tre blocchi funzionali: il Resource Discovery, il Front-end e il Resource and QoS Manager. Il Resource Discovery raccoglie e indicizza le risorse. Il Front-end raccoglie le richieste degli utenti C@H e li invia al modulo appropriato. La Resource and QoS Manager(RQM) gestisce e coordina le componenti del C@H, anche tiene sotto controllo lo stato del sistema e delle richieste.

Le iterazioni tra utente e provider sono governate norme di protocollo WS-Agreement [32] definie all'interno del SLA Manager. Il processo di negoziazione è supportato da CHASE per mezzo di uno strumento di simulazione e previsione.

Sulla base della previsione CHASE, il SLA Manager decide di accettare le proposte

del utente C@H o di respingerle. In caso di rifiuto, l'utente C@H può emettere una nuova proposta, nella forma di una contro offerta con meno esigente di parametri di QoS, cosa che il sistema potrebbe probabilmente sostenere. Al termine di una trattativa di successo, un contratto di servizio è firmato da entrambe le parti.
Dopo la negoziazione, il controllo viene consegnato alla componente RQM. In questa fase il RQM, in qualità di Service Manager, raggiunge la prestazione di servizio effettivo, vale a dire, si occupa di ricerca e di attivare le risorse che dovranno supportare i requisiti funzionali e non funzionali per l'utente C@H, espresse in SLA. Dopo di che, il RQM invoca un servizio per creare una infrastruttura ad hoc dedicata esclusivamente al monitoraggio delle risorse.
La componente MAGDA effettua il monitoraggio delle risorse assegnate. MAGDA fornisce un servizio in grado di eseguire diverse forme di monitoraggio, come un campionamento di dati locali (utilizzo corrente della CPU, la memoria disponibile, etc.) o come il monitoraggio distribuito attuato attraverso una piattaforma di agenti mobili. Inoltre, gli agenti mobili sono in grado di aggiornare un archivio di misure raccolte, al fine di effettuare l'analisi dei dati storici. È importante sottolineare che MAGDA sulle VM distribuite, consumando una frazione molto piccola della loro memoria e potenza di calcolo.
Ogni volta che l'infrastruttura di controllo rileva una possibile violazione del SLA, il RQM viene allertato e sollecitato a reagire contro il degrado QoS. Il RQM elabora le opportune contromisure (riconfigurare, riallocazione, esegue la migrazione delle istanze di VM). In questo contesto, RQM può invocare di nuovo CHASE al fine di raccogliere nuove stime per la selezione di nuove risorse disponibili per la migrazione delle VM, se necessario.
Infine, quando il servizio si completa in accordo alla SLA, o l'azione intrapresa deve essere forzatamente interrotta a causa della impossibilità di recupero o da un guasto, si invoca la fase di terminazione dal RQM.

Per causa della sua natura una nuvola non sarà mai in grado di fornire servizi affidabili ai consumatori, e la disponibilità delle risorse non può essere mai garantita. Questo modello aggrega spontaneamente le risorse, e fornisce un ulteriore livello su

di lore che si occupa di garantire un elevato livelli di disponibilità delle risorse. Il sistema è dotato di funzionalità e servizi per la negoziare del livello desiderato di QoS in forma di disponibilità delle risorse.

4.2 Modello basato su MTTF e MTTR

In questa sessione si introduce un modello runtime pronostico [33] sulla disponibilità di servizi software come elemento di QoS. Il modello si basa sulla previsione di due misure di disponibilità del servizio: *il tempo medio al guasto* (mean time to failure, MTTF) e *il tempo medio di riparazione (mean time to repair, MTTR)* di un servizio software.
In questo modello il MTTR di un servizio software è definito come il tempo medio da quando si verifica un guasto ad un servizio a quando il servizio riprende a rispondere alle chiamate correttamente. MTTR deve essere limitato per garantire la tempestiva riattivazione di un servizio dopo un periodi che il servizio non è disponibile. Quindi in un SLA, questo sarebbe in genere indicato come un vincolo MTTR $\leq$ K dove K è un'unità costante di tempo.
La probabilità di violare il vincolo MTTR$\leq$ K in un punto futuro t_e si basa sull'identificazione delle funzioni di distribuzione di probabilità di due variabili:

- MTTR del servizio.
- il tempo tra chiamate a servizi non servite che si verificano in un periodo in cui il servizio è stato disponibile (indicato come tempo di guasto o "TTF).

Valori MTTR e TTF corrispondono ai periodi indicati nella figura 4.3. Più in particolare, MTTR è calcolato come media dei valori di TTR, vale a dire, la differenza di tempo della prima chiamata di servizio completata dopo un periodo di indisponibilità e l'iniziale chiamata non servita (NS chiamata) del servizio che ha avviato questo periodo. TTF è la differenza tra i timestamp delle due chiamate NS dei servizi che avviano due periodi distinti e successivi di indisponibilità.

La probabilità di violare il vincolo MTTR$\leq$K al termine del periodo di tempo p può essere stimato approssimativamente con la formula seguente:

$$Pr\left(\bigwedge_{y=1}^{M} E_y\right) = \begin{cases} 1 - \sum_{y=1}^{M} Pr(y) * Pr(MTTR_y \leq MTTR_{crit}), & \text{MTTR}_\text{c} > \text{K}, \\ \sum_{y=1}^{M} Pr(y) * Pr(MTTR_y > MTTR_{crit}), & \text{MTTR}_\text{c} \leq \text{K} \end{cases}$$

La formula sopra distingue due casi:

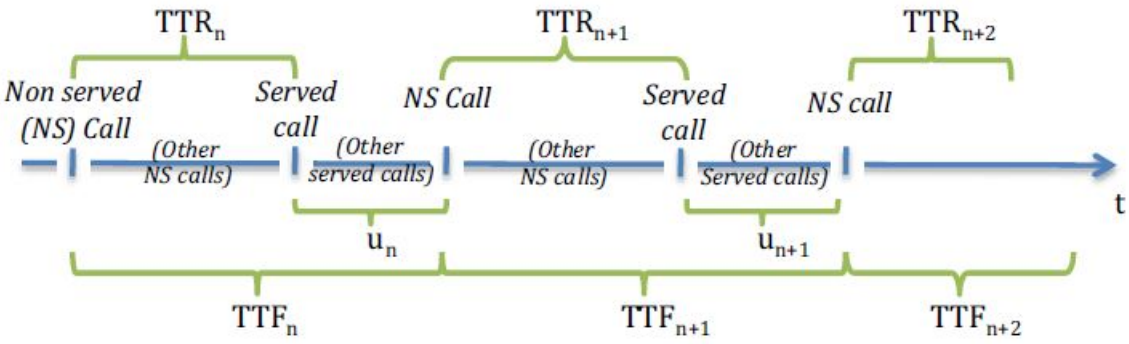

Figura 4.3: TTR e TTF.

- il caso in cui l'ultimo valore MTTR registrato quando viene richiesta la previsione, viola il vincolo (cioè, il caso in cui $\text{MTTR}_c > \text{K}$),
- il caso in cui l'ultimo valore MTTR registrato quando viene richiesta la previsione, non viola il vincolo (cioè, il caso quando $\text{MTTR}_c \leq \text{K}$).

Nel primo caso, la probabilità di violazione viene calcolata come la probabilità di non vedere un valore di MTTR nel periodo p (cioè, MTTR_y) che sia sufficientemente piccolo e per eleminare la violazione corrente. Nel secondo caso, la probabilità della violazione viene calcolata come la probabilità di vedere un valore sufficientemente grande MTTR_y (cioè, un valore maggiore di MTTR_{crit}) che violerebbe il vincolo. Nel caso del MTTF, il vincolo SLA da monitorare è $\text{MTTF} \geq \text{K}$ (quanto più grande il valore del MTTF meno frequenti saranno i fallimenti del servizio fornito) e la probabilità di violare un vincola può essere calcolata approssimativamente con:

$$Pr\left(\bigwedge_{y=1}^{M} E'_y\right) = \begin{cases} 1 - \sum_{y=1}^{M} Pr(y) * Pr(MTTR_y \geq MTTR_{crit}), & \text{MTTR}_c < \text{K}, \\ \sum_{y=1}^{M} Pr(y) * Pr(MTTR_y < MTTR_{crit}), & \text{MTTR}_c \geq \text{K} \end{cases}$$

Questo modello presentato un approccio black-box per predire la disponibilità dei servizi software sulla base delle previsioni del MTTR e MTTF dei servizi stessi. Il modello soddisfa i requisti minimi di disponibilità e la garantisce per tutta la durata del servizio.

Capitolo 5

Affidabilità

Nonostante i notevoli vantaggi offerti, molte aziende hanno ancora dubbi sull'adozione del cloud computing. Il principale motivo di esitazione è legato all'affidabilità, con tutti i problemi di sicurezza e conformità che ne derivano. Ciò è confermato da uno studio condotto da Gartner nel 2009, secondo il quale i sei principali ostacoli all'adozione del cloud riguardano l'affidabilità. I maggiori timori vanno dalla protezione dei segreti commerciali businesscritical al rispetto dei requisiti legali, fino alla sicurezza delle informazioni personali.
Naturalmente le aziende e i loro reparti IT si trovano oggi a dover affrontare le stesse sfide, che però si manifestano diversamente con il passaggio al cloud: grazie a questo approccio, le organizzazioni possono risolvere le problematiche e modificare di conseguenza le strategie di implementazione del cloud adottate.
L'affidabilità è essenziale per tutte le distribuzioni cloud. Al fine di sostenere le applicazioni dei data center in un nuvola, l'affidabilità è considerata una delle caratteristiche principali da sfruttare. L'affidabilità denota la capacità di garantire un funzionamento costante del sistema senza interruzioni, cioè senza perdita di dati, nessun reset di codice durante l'esecuzione ecc. L'affidabilità è tipicamente raggiunta mantenendo un certo livello di ridondanza delle risorse [35] .
L'affidabilità è un aspetto particolare della QoS che deve essere specificato per garantire ala qualità dei servizi.

È interessante notare che tra affidabilità e disponibilità esiste una relazione

molto stretta, comunqe la differenza sta nel fatto che l'obbiettivo dell'affidibilità è la prevenzione delle perdite (i.e:dati).

In questo capitolo verranno introdotti e commentati alcuni modelli di rilevanza che trattano alcuni aspetti fondamentali per garantire l'affidabilità come elemento della QoS.

5.1 S^5

Il modello presentoto in questa sezione e un medolle composto da cinque strati è chiamato S^5 (SEE, SAVE, SCAN, SHALL, SAFE) [36]. Questi cinque strati insieme sono in grado di gestire tutte gli attributi di affidabilità relativi ai servizi cloud. Inoltre, la prevenzione ai difetti, a differenza di approcci tradizionali, si tiene una traccia dei guasti e si fanno previsione per garantire l'affidabilità. Infatti il sistema quanti ogni volta che determina un guasto e lo anienta, aumenta la sua affidabilità.

Le definizioni di QoS fin'ora fornite non offrono una visione dell'affidabilità al quanto sfumata. In [36] si introduce un nuovo concetto di qualita di affidabilità, Quality of Reliability (QoR) e i suoi requisiti per i sistemi di servizi cloud. Siccome i servizi sono realizzati per soddisfare le necessità umane ci si può basare sulla gerarchia dei bisogni do Maslow[1] per determinare i requisiti di affidabilità. Sulla base di questa toeria le richieste vengono divise in base alla gerarchia dei bisogni umani (bisogni fisiologici, di sicurezza, d'appartenenza, di stima e di auto-realizzazione), possono essere definiti quattro livelli di requisiti per la QoR:

1. Esistenza di servizio: L'esistenza di un servizio fonda il livello fondamentale in S^5.

2. La disponibilità del servizio: Dal momento che i servizi cloud devono collaborare, si esige un livello di disponibilità del servizio per garantire la stabilità del servizio può migliorare la percezione del servizio per i consumatori.

3. Capacità di servizio e l'usabilità: L'efficienza di esecuzione e la precisione sono anche di vitale importanza per il QoR. Pertanto, il compito principale del terzo livello è quello di garantire la funzionalità e la fruibilità delle componenti di servizio.

[1] Nel 1954 lo psicologo Abraham Maslow propose un modello motivazionale dello sviluppo umano basato su una "gerarchia di bisogni", cioè una serie di "bisogni" disposti gerarchicamente in base alla quale la soddisfazione dei bisogni più elementari è la condizione per fare emergere i bisogni di ordine superiore.

4. Servizio di self-healing: Un strato di self-healing è necessario nel caso di alcuni eventi critici, secondo questo il framework S^5 dovrebbe essere in grado di ripristinare alcune componenti di sistema del servizio cloud.

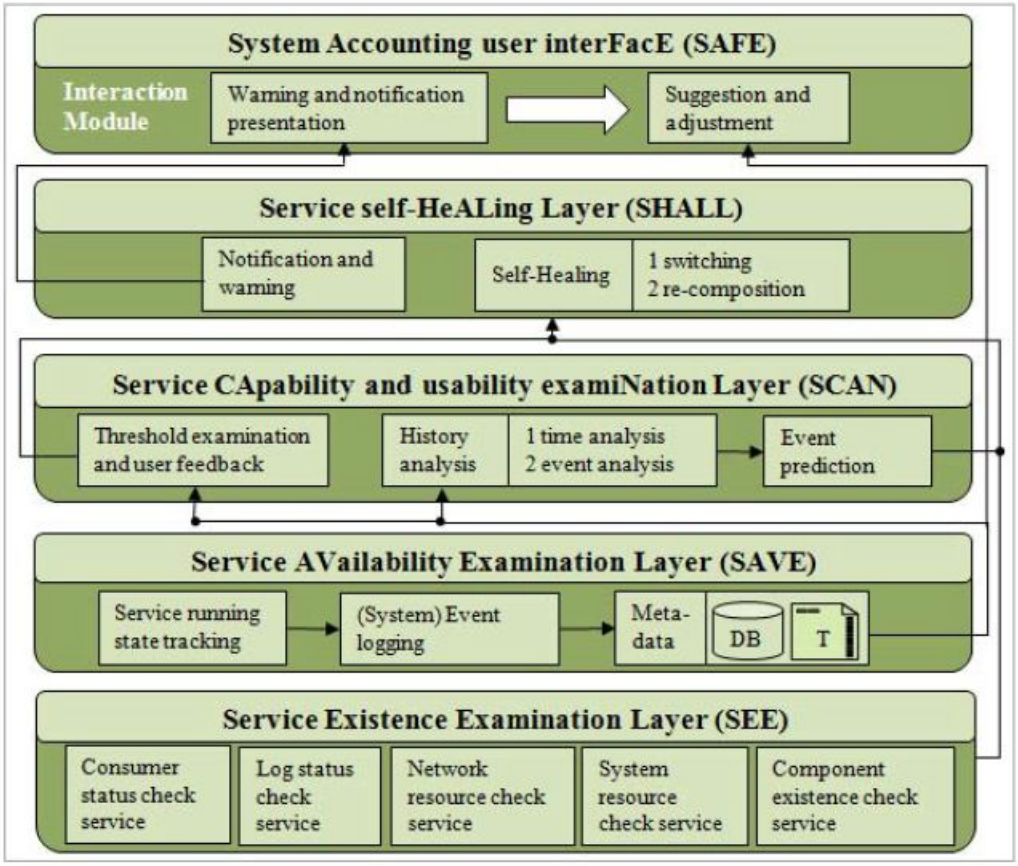

Figura 5.1: Architettura S^5.

L'obbiettivo di questo modello è quello di soddisfare tutti i requisiti per i sistemi di servizi, integrare le attività dei fornitori e dei consumatori nei sistemi cloud, migliorare le prestazioni dei servizi, e quindi garantire i bisogni dei consumatori. L'architettura S^5 è schematizzato in figura 5.1.

Service Existence Examination (SEE): Dato che l'esistenza dei servizi è il presupposto di base per tutti i fornitori e i consumatori, la funzionalità principale di questo livello è quello di esaminare, in tempo reale, l'esistenza di ogni sistema o componente di servizio nei sistemi cloud. Questo strato contiene di diversi moduli,

come mostrato in figura 5.1. Una volta che si determina che il servizo non esiste, informazioni dettagliate sono trasmesse al livello di Service Self-Healing (SHALL), in modo che questo strato notifichi i consumatori ed esegua un protocollo di guarigione o recupero. Nel livello più fondamentale di servizi cloud, l'efficacia di esecuzione è la prima considerazione.

Service AVailability Examination (SAVE): Lo strato SAVE, determina se le componenti dei sevizi funzionano e se il loro stato è normale. Basandosi sulla salute delle componenti di sistema e di servizio, il modulo "Service Running State Tracking (SRST)" tiene traccia dello stato di esecuzione di ciascuna componente di servizio. Nel frattempo, secondo un livelli di evento predefinito, i risultati del monitoraggio sono gestiti dal modulo "(System) Event Logging (SEL)" per differenziare gli eventi ricevuti. Qui, SEL elabora gli eventi generati dai sistemi operativi, i quali sono utilizzati dai sistemi di servizi cloud. In SAVE, i log vengono registrati in un database sul lato provider, e anche in un insieme di file di testo sulle macchine dei consumatori, per ridondanza e motivi di backup.

Service CApability and usability examiNation (SCAN): Anche se, sulla base di SAVE, la disponibilità del servizio può essere garantita fino ad un certo livello, non esiste un metodo appropriato per soddisfare i requisiti di un livello superiore di QoR. Diventa necessario analizzare lo storico dell'esecuzione. Qua interviene SCAN che offre anche elementi per raffinare la QoR, questo eseguito in quattro punti.

- *Analisi degli eventi:* In questo framework del S^5, si analizano gli eventi con l'obbiettivo di scoprire le regole o le probabilità con le quali essi occorreranno. Per conoscere le probabilità che si verifichi un determinato evento, vengono utilizzati approcci come Pipelining Aho-Corasick (P-AC)[37] e Averaged One-Dependence Estimators (Aode) [38].

- *Previsione Evento:* Gli eventi o i log pattern riconusciuti possono essere utilizzati per prevedere eventi anomali. Appena P-AC individua qualche irregolarità, questo modulo può scegliere l'evento con la massima probabilità

di anomalia come il più possibile anomalia a verificarsi. Il prossimo evento sarà contemporaneamente trasmesso come un messaggio urgente al livello SAFE.

- *Esaminazione dei threadshold e dei feedback degli utenti:* In SCAN, il threadshold viene esaminato per ogni indicatore, come ad esempio l'utilizzo della CPU, l'utilizzo della memoria, il tempo di risposta del sistema, il tempo di risposta dei consumatori, ecc.

Service self-HeALing Layer (SHALL): questo livello del S^5, per i componenti del servizio o del sistema che falliscono il sistema eseguirà un processo di ricerca per trovare un insieme di unità di backup con la funzionalità del servizio stesso o simile. Self-healing è una funzionalità on-demand, il che significa che l'azione di guarigione viene attivata da eventi di livello inferiore, come gli errori di esistenza, problemi di disponibilità, e problemi di capacità. Due moduli guarigione sono forniti in questo strato.

- *Modulo di commutazione:* Cerca un servizio di backup nello stesso provider per fare una sostituzione.

- *Modulo di ricomposizione:* Al fine di costruire una servizi come combinazione di componenti più piccole[2], un approccio modificato di Hierarchical Task Networks (HTN) viene utilizzato per comporre un nuovo servizio con le migliori componenti disponibili.

System Accounting user interFacE(SAFE): Ogni componente del servizio in sistemi di servizi di cloud è un nodo in una catena di servizi o in un albero. Lo strato SAFE funziona su ogni consumatore di ricevere avvisi e notifiche, e per fornire un insieme di interfacce utente per gli amministratori di sistema di regolazione le corrispondenti soglie di sistema entro i valori consentiti.

In questo modello, secondo le disposizioni di QoR per i servizi cloud, il framework S^5 determina l'esistenza dei servizi, esegue analisi sulla storia dei servizi, monitoraggio in tempo reale e la previsione sugli eventi in modo da garantire una massima affidabilità dei servizi cloud.

[2]Utile in caso di guasti, permetta la sostituzione del pezzo guasto.

5.2 Magicube

Sistemi di storage cloud come Google File System (GFS) [47], Hadoop Distributed File System (HDFS) [10] e Amazon's Simple Storage Service (S3) [11] sono sistemi complessi composti da un elevato numero di componenti, in cui si possono verificare con ferquenza guasti quali problemi hardware, bug software e partizione della rete. Per rendere questi sistemi più affidabili, molte azioni sono state intraprese per affrontare tutti i tipi di guasti. Il metodo più diffuso per rispondere all'affidabilità dei file system distribuiti è la replicazione di tali file. La maggior parte dei più popolari sistemi di storage distribuiti, utilizzano questa politica. Ad esempio, e in GFS, HDFS e Amazon S3 vengono mantenute tre copie dei file.
Nonostante l'alto livello di affidabilità che si ottiene mantenendo multiple repliche dei file, si introduce alto overhead in spazio di memoria. Ad esempio, in un sistema che mantiene tre copie, per 1GB di dati validi si consumeranno 3 GB di spazio di archiviazione, e il sovraccarico è 2 GB, l'effettivo spazio utilizzoto è solo il 33%. Più repliche si usano più spazio viene consumato.
Magicube [39] è una architettura di di storage che fa un compromesso tra alta affidabilità e basso overhead di spazio per i sistemi di cloud storage. Infatti,e logico dire che l'alta affidabilità e uno spazio ridotto sono due cose sempre contraddittorie nei sistemi di storage esistenti.
Per ridurre l'overhead di spazio di archiviazione dei file, Magicube mantiene una sola copia per ogni file in HDFS, e per ottenere alta affidabilità, Magicube utilizza uno speciale algoritmo di codifica per fault-tolerant. La caratteristica fondamentale della codifica Magicube è che, una volta che i parametri di sistema *(n, k)* sono stati impostati, il file di origine viene codificato e il file cifrato sarà diviso in n parti, ciascuna salvata nel cluster. Se il file sorgente viene perso, Magicube è in grado di ripristinare il file con un delle *k (k <n)* su n parti del file codificato. Inoltre, Magicube permette agli utenti di regolare il livello di tolleranza agli errori e costo dello spazio modificando le impostazioni di (n, k).

La figura 5.2 mostra l'architettura di Magicube. Il sistema è costruito in cima a HDFS. L'utilizzo di HDFS e rivolto alle sue funzioni di per lettura
scrittura di file e la gestione dei metadati. Inoltre, la componente di riparazione dei

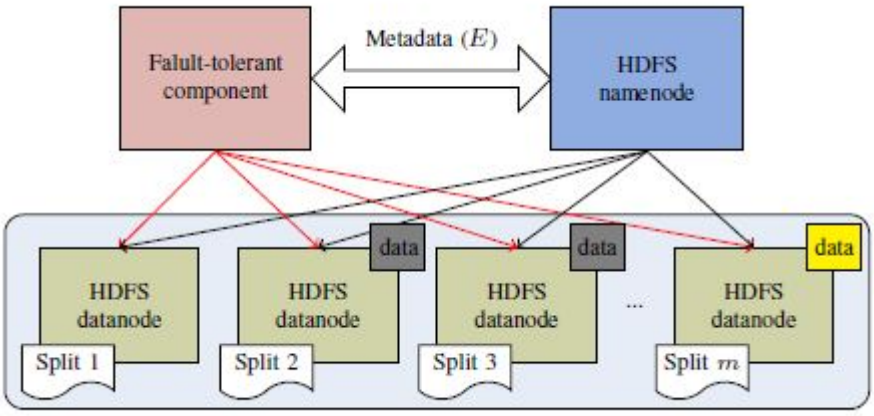

Figura 5.2: Architettura Magicube.

file, sopra citata, opera in sottofondo in modo indipendente da HDFS. Il modulo di Fault-tolerant e HDFS sono due parti distinte in questo sistema, comunque comunicano tra di loro durante la divisione e distribuzione dei file e nell'accesso alle k parti di file di riparatite. Per ottenere affidabilità, basso costo per lo spazio di memoria e elevate prestazioni, il sistema lavora nel modo seguente:

- *Triplicazione all'inizio:* solitamente HDFS serve per la lettura
  scrittura dei file. Per ottenere elevate prestazioni e alta affidabilità, all'inizio dell'esecuzione di un'applicazione client si creano e mantengono tre copie dei dati in HDFS.

- *Suddivisione dei file e distribuzione:* non appena si completa la scrittura su HDFS, si attiva in background il modulo Fault-tolerant per la scissione dei file. Subito dopo il HDFS è responsabile per la distribuzione di questi file nel cluster in modo più uniforme possibile.
  Nel frattempo, HDFS mantiene i metadati di base della suddivisione, riparazione, e l'ubicazione dei file in una matrice E.
  Per fare un uso migliore di spazio su disco, HDFS può scegliere anche solo m *(k <m <n)* su n parti di file da memorizzare nel cluster.

- *Eliminazione delle repliche eccessive:* qando si completa la divisione, distribuzione, salvataggio dei file, verranno eliminate due su tre repliche

presenti in HDFS. Le repliche da eliminare vengono scelte in base alla distribuzione del carico su cluster.

- *File Repair:* la riparazione dei file si esegue soltanto quando la copia originale del file ubicata in HDFS non è più disponibile. Quando questo accade, il modulo Fault-tolerant fara richiesta dei pezi di file salvati nel cluster. Quando ottiene k parti su n, il modulo Fault-tolerant è in grado di ricustruire il file originale.

Magicube è un modello affidabile e risparmia spazio nei cloud storage. Gli esperimenti svolti con Magicube garantiscono il funzionamento. Unico problema che tale sistema può avere è che nel tenatativo di garantire un alta affidabilità aggiunge una componente di computazionale nella faze di codifica e ripartizione dei file. Tale aggiunta ha effotto economico siccome si prevede che il modulo per la ripartizione venga posto nel sistm in maniera disgiunta.

Capitolo 6

Sicurezza

Il cloud Computing presenta molti rischi per la sicurezza e problematiche che riguardano la privacy. In tale ambiente, i confini del sistema, che erano delimitati, diventano sempre più ampi. Gli utenti, aziende che ripongono le proprie informazione e risorse nel nei sistemi cloud sono continuamente preoccupati per la vulnerabilità del sistema riguardanti eventuali attachi. Pertanto, eventuali attacchi possono causare problemi che riguardano la disponibilità e la capacità del sistema di soddisfare la richiesta degli utenti del cloud in base ai loro contratti SLA e rispetto ai requisiti di QoS. Così, aspetti di sicurezza da considerare e affrontare sono la riservatezza, l'integrità dei dati, controllo e audit.

La nube ha molte caratteristiche, come la quota di risorse, l'elasticità e la virtualizzazione, che possono presentare diversi rischi per la sicurezza. Infatti, l'uso della stessa istanza di servizio da molti utenti pone problemi di sicurezza e di controllo soprattutto a livello hardware. Ecco perchè le risorse condivisibili devono essere ben isolata e protetta per per non esporre gli utenti ad rischi generati da altri utenti. Inoltre, l'elasticità, che consente una allocazione dinamica delle risorse secondo necessità dell'utente, può generare l'utilizzo di risorse che sono già state attribuiti ad altri utenti. La virtualizzazione dei sistemi elimina confini fisici, macchine virtuali possono essere copiate o clonate. Per ridurre l'impatto della virtualizzazione in argumenti di sicurezza, i sistemi di accesso devono essere controllati e garantiti in modo sicuro. Queste vulnerabilità possono essere sfruttate da un hacker per compromettere il servizio o la riservatezza. Così, molti attacchi possono verificarsi

sul cloud computing.

6.1 Rischi Cloud

Le minacce esterne sono uno dei piu' grandi problemi relativi a qualsiasi organizzazione in quanto, possono compromettere direttamente informazioni riservate o deturpare un organizzazione. Questo è uno dei problemi persistenti in infrastruttura Cloud pure poichè il cloud è user friendly, rispetto alle reti private. Inoltre il cloud dispone di più interfacce per aiutare i legittimi utenti a recuperare le proprie informazioni. Questo è ciò che i hacker utilizza a suo vantaggio facendo leva sulle debolezze dell'API, della connessione (media o canale logico) toccando o irrompendo in un social engine.

Direct DoS attack mira ad impedire al legittimo utente di accedere a informazioni o servizi. Essa si verifica quando un utente malintenzionato invia un gran numero di richieste senza senso a un determinato servizio. Un altro attacco contro la disponibilità è *indirect DoS*. Sappiamo che una delle caratteristiche comuni del sistema cloud è di fornire risorse dinamicamente scalabili. Quando un utente malintenzionato esegue un attacco DoS contro un particolare servizio nel sistema, il cloud inizia a fornire più istanze di servizio per far fronte al carico di lavoro. Alla fine, il sistema potrebbe consumare tutte le risorse che il server è in grado di fornire, ciò rende all'utente impossibile accedere ad altri servizi. In più gli altri servizi in esecuzione sullo stesso hardware potrebberò soffrire il carico di lavoro causato da un attacco DoS. Anche se è difficile da prevenire completamente il direct/indirect DoS attack, un firewall o un instruction detect system è in grado di filtrare le richieste dannose.

L'attacco *Malware Cloud Injection* è un attacco contro l'integrità e la riservatezza. L'attacante tenta di iniettare un servizio dannoso, un applicazione o un computer virtuale nel cloud a seconda del modello di servizio cloud (SaaS, PaaS o IaaS). Per poter eseguire questo attacco, l'attaccante deve creare il suo proprio servizio, applicazione o istanza di macchina virtuale e successivamente aggiungerla al Cloud. L'attacante deve ingannare il sistema facendogli credere che l'istanza maligna aggiunta sia nociva. Se succede, le richieste valide possono essere reindirizzate all'istanza maligna, che conclude con l'esecuzione del codice dannoso.

Un altro scenario del tipo di attacco sopra esposto, potrebbe essere che un utente

malintenzionato tenta di caricare un virus o un programma Trojan al sistema cloud. Possibile contromisura per questo tipo di attacco è eseguire una istanza per l'integrità che controlla le instanze di servizio prima che vengano utilizzate da richieste in ingresso.

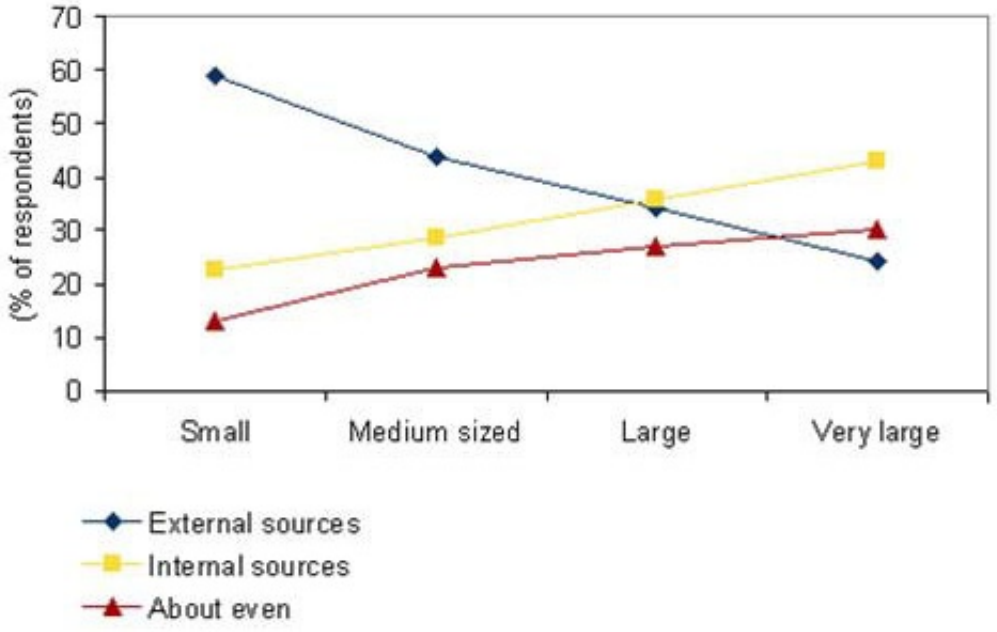

Figura 6.1: Rischi interni vs. rischi esterni.

Tante minacce rivolte alla sicurezza derivano dall'interno di un'organizzazione. Questa minaccia è rivolta a molti consumatori di servizi cloud in quanto, il sistema si basa sul modello multi-tenant, sotto la gestione di un singolo provider. Come se non bastasse, le organizzazioni che sottoscrivono ai servizi cloud, di solito mancano di trasparenza nel processi di assunzione dei dipendenti del provider, di conservazione dei dati in luoghi diversi e le sue relazioni con i fornitori di terze parti. Spesso non c'è visibilità sulle norme e le pratiche di assunzione per i dipendenti nuvola, dal provider di cloud verso i suoi clienti. Questa fatto può fare spazio ad un avversario da un punto di vista di spionaggio industriale, hacker casual o addetti ai lavori dannosi. La seguente figura 6.1 [40] riferisce dettagli risultanti da un'indagine condotta dall'IDC per riflettere il minacce interne che esiste in tutto il mondo dalle piccole alle grandi.

6.2 Modelli per garantire la sicurezza

In questo capitolo verranno introdotti e commentati alcuni modelli di rilevanza che trattano alcuni aspetti fondamentali per garantire la sicurezza come elemento della QoS. Per quanto riguard la sicurezza è difficile trovare materiale che la tratta come elemento di QoS e modelli implementati per garantire la QoS rivorlti esplicitamente ai problemi riguardanti questo elemento.

6.2.1 UBIS per la QoS

UBIS(Ubiquity and Integration of Services)[45] è un modello che combina due approcci, SOA (Service Oriented Architecture) e EDA (Event Driven componenti del servizio Architettura), per rispondere alle sfide di sicurezza e QoS nelle reti Cloud. Questa architettura assicura al cloud una composizione sicura di servizi in modo dinamico e consistente.
In questa architettura, il servizio è considerato come un'unità atomica e ogni servizio rappresenta un feature ben definita per gli attori umani o per altri servizi. I cloud UBIS offrono servizi specifici e gli utenti UBIS possono consumare questi servizi all'interno della rete. I fornitori dei servizi UBIS hanno la responsabilità di garantire la qualità, le prestazioni, l'affidabilità e la sicurezza dei servizi offerti. In realtà, gli utenti cloud sono ingordi e pretendono di accedere ai propri servizi senza oblighi e vincoli di tipo temporali, geografiche, tecnici o economici.

Pertanto, UBIS Cloud Service (UCS) mira a stabilire una sessione dinamica che meglio si adatta alle loro preferenze e alle loro esigenze di QoS. Questa architettura tenta di combinare i servizi offrendo interoperabilità, l'interazione e la cooperazione tra i diversi componenti di servizi. Vengono rispettati sia aspetti funzionali e non funzionali per offrire un servizio globale con requisiti di QoS garantiti.
Durante l'esecuzione di una sessione, da parte di un utente, ogni componente di servizio è autogestito. Questo permette di garantire l'automazione e decentralizzazione del controllo e gestione dei servizi. In effetti, è stato integrato, in ogni componente di servizio, un agente che controlla per ogni componente di servizio lo stato corrente della QoS, verificando se soddisfare la qualità del servizio (partendo dal SLA) e i requisiti di sicurezza. Così, quando la QoS degrada o si verifica un

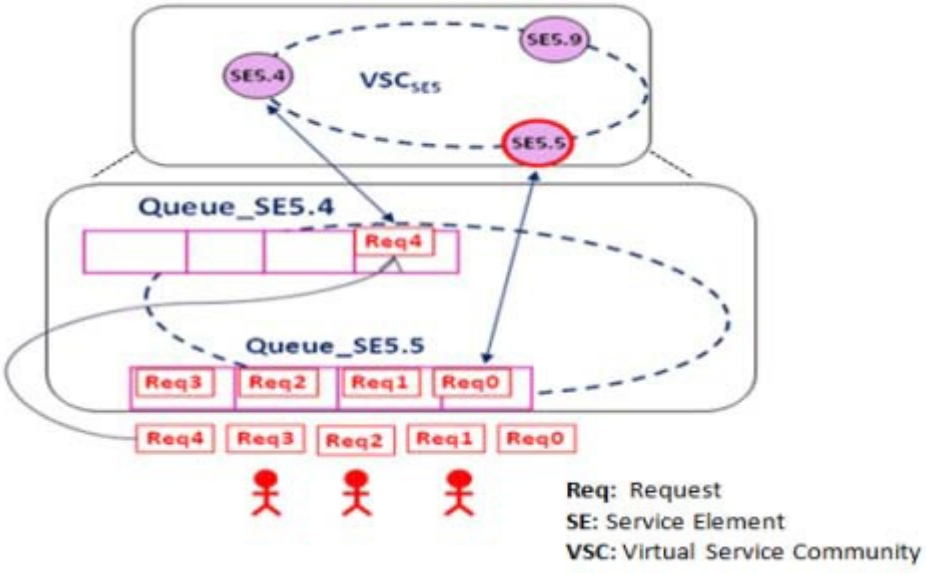

Figura 6.2: Attacco DoS.

attacco alla sicurezza un si verifica e si genera un evento, la componente affetta sarà sostituita nella sessione attiva con un altra componente di servizio ubiqua. La funzione ubiqua viene gestita con un modello di sicurezza chiamato VSC (Virtual Community Service). Ogni componente di servizio in una sessione è parte di un VSC che ragruppa un insieme di funzionalmente e componenti di QoS equivalenti. Sono stati scelti quattro criteri per descrivere le caratteristiche di servizio in termini di QoS. Queste caratteristiche sono le seguenti:

- Disponibilità/Accessibilità: rappresenta il tasso di richieste non accettate dal componente del servizio. Per massimizzare la disponibilità di una componente di servizio, è stato distribuito un numero di componenti ubique per coprire la domanda (dimensionamento), e aggiunte alcune code per garantire SLA ad ogni query.

- Affidabilità: rappresenta il tasso di errore di richieste indirizzate alla componente di servizio.

- Capacità: rappresenta la potenza media di calcolo per unità di tempo.

- Ritardo: il tempo medio di calcolo.

Consideriamo il caso di superamento della soglia di tempo disponibile per un componente del servizio. Ad esempio, questo può essere causato dalla scadenza del credito dell'utente. L'autorizzazione per accedere a questa componente non sarà più valida. Di conseguenza, questo componente viene sostituita da un altro servizio ubiquo.

Inoltre, molti tentativi di accesso, falliti e consecutivi, ad una componente di servizio può rendere il medesimo servizio non disponibile ed essere interpretato come un attacco. Supponendo che un hacker tenta di rendere il servizio SE5.5, in figura 6.2, per interrompere la sua disponibile. Egli manda centinaia di richieste nella coda di SE5.5. Quando questa coda è piena, il suo agente QoS reindirizza le richieste successive ad un'altra coda ubiqua che appartiene alla componente SE5.4. Attraverso questo meccanismo, il carico per un servizio è distribuito su componenti ubiquitari che offrono la stessa QoS, in modo da ridurre notevolmente l'effetto di attacco DoS. Questo modello infatti è principalemnte indirizzato a prevenire attacchi maligni. La QoS per la sicurezza viene sondata in base alle prestazioni del sistema, nel momento che il sistema non risponde secondo i quattro componenti di servizio proposti nel modello, si pone una situazione che deve essere gestita per rigarantire la sicurezza e la QoS, e' fare in modo che l'utente proprietario del servizio riesca ad accederci.

Problema che si nota nel sistema e' che avendo le componenti ubique, un eventuale attacco non compromette soltanto un utente ma anche altri consumatori del UCS.

6.2.2 Sicurezza in UCaaS

Unified Communication (UC) è comunemente inteso come l'integrazione dei servizi di comunicazione real-time e non real-time, consegnati al cliente in modo uniforme e integrato. UC attualmente sta migrando a un servizio fornito mediante una infrastruttura Cloud. In quest'ultimo caso, si parla di UC-as-a-Service (UCaaS). Dal punto di vista del cliente, il vantaggio principale di UCaaS rispetto alla tradizionale distribuzione UC è il modello pay-per-use, in cui i clienti pagano solo per i servizi di cui hanno bisogno e la durata effettiva dell'utilizzo. Questo modello è quindi particolarmente interessante per le piccole medie imprese, che cercano di evitare grandi investimenti. UCaaS offrire anche la possibilità di avere a disposizione diversi servizi per i clienti diversi, e pure il supporto per l'aggiunta e la rimozione di servizi on-demand.

Questo modello[46] prende spunto dalla definizione e il ciclo di vita del SLA. Il SLA per la sicurezza ha un ciclo di vita che si concentra sulle interazioni tra il cliente e il fornitore del servizio. Qui si descrivono i passi necessari nel ciclo di vita. Come si vede in figura 6.3, è costituito da sei differenti fasi; pubblicazione, negoziazione, impegno, provisioning, il monitoraggio e la terminazione.

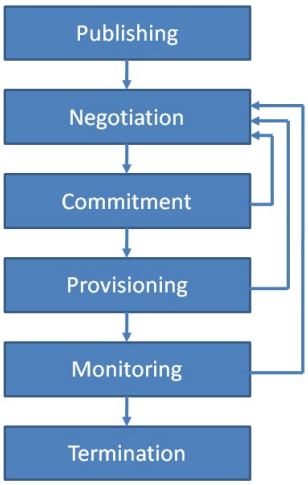

Figura 6.3: Il ciclo di vita del SLA.

La prima fase comprende la creazione e la pubblicazione di offerte di servizio

in un catalogo di servizi, in modo trovare potenziali nuovi clienti. In questa fase i provider progetta e prepara i modelli di SLA e di sicurezza in base alle proprie capacità , strategie di business e le relazioni con gli sviluppatori di servizi. Un modello di SLA pubblicato deve come minimo contenere i meccanismi di sicurezza offerti e i costi relativi. Inoltre, il periodo di validità deve essere indicato in un modello di SLA pubblicato.

Nella fase di negoziazione il cliente e il provider devono concordare i dettagli di sicurezza per il SLA. Questa fase è ulteriormente illustrata in figura 6.4. In questo modello la fase di negoziazione è composta da varie sottofasi. Una trattativa può essere eseguita da agenti software che agiscono per conto del cliente e fornitore. Come si può vedere nella figura 6.4, il cliente invia una lista di requisiti di sicurezza per il fornitore del servizio, invece il provider fa le sue proposte. Questo modello può essere implementato anche in cloud ibride, e possono esserci più fornitori di servizio. La fase di negoziazione si tradurrà in misure di sicurezza per il SLA tra il cliente e il fornitore del servizio, o tra il fornitore di servizi. Come si vede nella figura 6.4 il SLA sicurezza finale tra il cliente e il fornitore di servizi può essere il risultato di interazioni multiple tra un certo numero di altri fornitori.

Una fase di negoziazione conclusa con successo sarà seguita da una fase di presa di un impegno, in cui il SLA per la sicurezza sono firmati digitalmente da tutti i partner coinvolti. Per un servizio di cloud ibrido il SLA di sicurezza risultante tra il cliente e il fornitore del servizio di solito è un risultato di catene multiple di contratti conclusi tra il fornitore di servizi e altri fornitori.

Il ciclo di vita proposto include anche una fase di approvigionamento, per configurare e realizzare i meccanismi di sicurezza concordati. Richieste di modifica in entrambi questa fase o la fase di impegno può portare alla rinegoziazione di sicurezza SLA.

La fase di monitoraggio viene utilizzata per garantire che i dettagli nella SLA

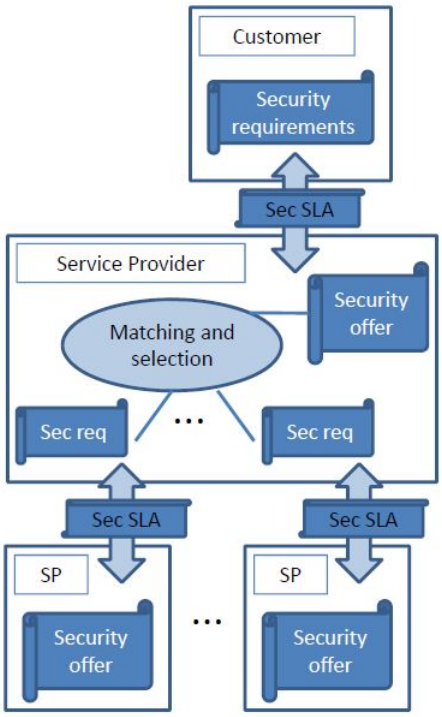

Figura 6.4: Negoziazione SLA.

sicurezza siano soddisfatte. Questa fase comprende il rilevamento di violazioni della sicurezza sia trascorsi che in corso, e interpretare se tali eventi si verificano per a causa di violazioni delle norme di sicurezza sottoscritte nel SLA. Qusto può portare alla rinegoziamento del servizio o alla cessazione.

La fase di terminazione chiude il contratto stipulato e sblocca risorse eventualmente riservati.

In questa sezione si mostra come si applica il ciclo di vita del SLA in un UC composto da un servizio VoIP (VOICE), un servizio di messaggi (MSG), un servizio di informazioni sulla presenza e un servizio di conferenza. Nella prima fase del ciclo di vita SLA per la sicurezza (figura 6.4) diversi provider pubblicano le loro offerte di sicurezza in conformità con il tipo di servizio offerti. Il cliente dovrà quindi confrontare l'offerta con la propria richiesta. In UCaaS le richieste prioritarie possono essere viste in tre livelli: DEVE (il cliente non accetta qualsiasi servizio che non

soddisfa questo requisito) , DOVREBBE (il cliente preferisce, ma può accontentarsi di meno) e PIACEVOLE (bello avere,ma non importante). Sulla base dei modelli di protezione pubblicati SLA, il provider di UCaaS confronterà i requisiti di sicurezza del cliente con le offerte di servizi e farà un breve elenco di tutti i fornitori in grado di rispettare tutte le prescrizioni.

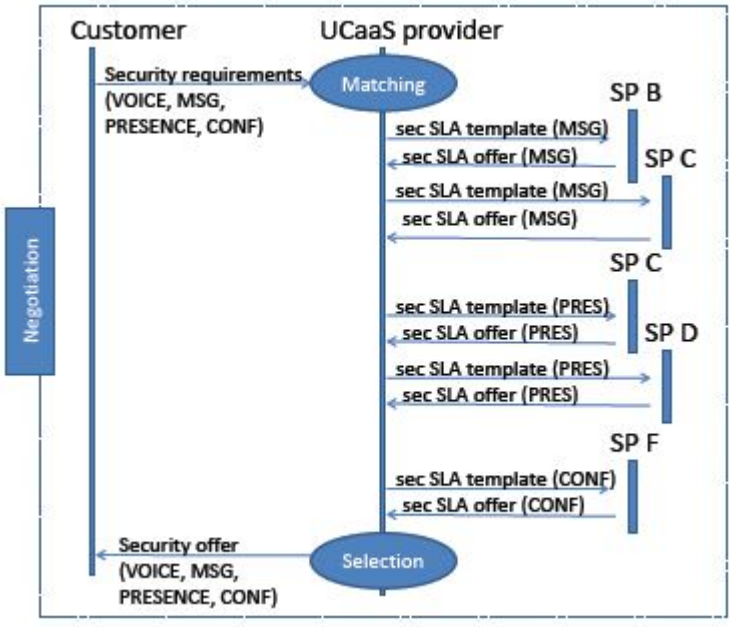

Figura 6.5: Flusso di messaggi nella fase di negoziazione.

In questo esempio specifico, il fornitore UCaaS ha un proprio servizio voce, e fino a quando questo è in grado di soddisfare tutti , il fornitore UCaaS non richiederà ulteriori provider esterni. Si assume che questo sia il caso in esempio. Le restanti componenti, il provider UCaaS avvierà una trattativa per ogni sotto-servizio, selezionando i requisiti rilevanti dalla lista in ogni caso. Il flusso di messaggi in fase di negoziazione è descritto nella Figura 6.5.

La sicurezza in Clouds ibride è una grande sfida. In UCaaS tutti i requisiti funzionali sono sempresoddisfatti dai sottoservizi. Questo accade raramente nella vita reale. Allo stesso modo, i requisiti di sicurezza potrebberocreare conflitto con altri requisiti di QoS in una fase di negoziazione del SLA.

Conclusioni

In questa tesi è stata prenentata una tecnoligia che e possiede un ampia prospettiva di evoluzione. Il cloud computing è un modello dirompente che ha cambiato il modo nel quale si usa l'IT a supporto delle persone e del bussines.
Però nonostante la sua larga distribuzione e elevato utilizzo non esiste uno standard di granzia per la sua QoS. Oggi i vincoli e il livello di servizio che viene offerto agli utenti viene definito come un accordo bilaterale tra utente e provider che si chiama Service Level Agreement (SLA). Molti approcci per garantire la qualità di servizio vengono offerti partendo dalla definizione del SLA e sua gestione.
In questa tesi si tenta studiare gli attributi di QoS in modo individuale e determinano gli aspetti fondamentali per ciascuno di essi.

In letteratura uno degli aspetti fondamentali e tra i più richiesti dagli utenti cloud sono le *prestazioni*. Infatti questo attributo è fondamentale per far fronte ad applicazioni che richiedono tante risorse di calcolo. Il problema principale che riguarda questo attributo deriva dall'architettura del cloud. Il cloud è impattato da come le applicazioni e le risorse vengono poste e distribuite nei cluster. L'approccio più comune è dato mediante meccanismi di monitoraggio continuo e algoritmi di distribuzione delle risorse, per non permettere che la prestazioni dei servizi non decadano sotto un certo livello partendo dal SLA.
Altro elemento della QoS, strettamente collegato con le prestazioni, è la *scalabilità*. La scalabilità in poche parole rappresenta la capacità del sistema a far fronte a un elevato numero di richieste improvvise, senza degradare la sua qualità. Solitamente la scalabilità si gestisce, come per le prestazioni, mediante meccanismi di monitoraggio. La scalabilità è un elemento che ha effetto immediato sulle prestazioni.

Un idea per il futuro, è fornire un meccanismo che gestisce questi problemi in modo lineare. L'idea è di offrire una divisione totale delle risorse del cluster in modo equilibrato tra gli utenti. Definire per ogni servizio una unita che funziona in background ma all'interno del servizio. Tale unità fa il monitoraggio continuo delle prestazioni e nel momento che determina una situazione di allarme effetua un broadcasts di S.O.S su tutti i servizi sul cluster e in caso un servizio disponesse di risorse inutilizzate le prende in "prestito". All'eventuale cessazione della situazione di rischio le risorse vengono ridistribuite. L'idea è quella di decentralizzare i meccanismi di monitoraggio e mediante un appropriato SLA, porli più vicine alle risorse degli utenti in modo da offrire un risposta immdeiata ai problemi.

Altri aspetti altrettanto fondamentali per la QoS sono l'affidabilità e disponibilità. Anch'esse sono strettamente collegate. È importante notare che l'affidabilità, a differenza della disponibilità, è rivolta a salvaguardare il sistema dalle perdite di dati. Metodi più comuni per afforntare questi due aspetti sono quelli di mantenere i dati duplicati però questo metodo è costoso, richiede l'impegno di tante risorse.
Altro metodo altrettanto interessante è quello di utilizzare modelli di previsione probablistici i quali in base al trascorso dei servizi tentano di gestire i requisiti di sistema per garantire la loro affidabilità e disponibilità.

Ultimo aspetto trattato è la sicurezza. Essa rappresenta ancora un grande punto interrogativo per gli utenti, il qaule li lascia incerti se devono migrare verso il cloud. Oggi giorno i problemi fondamentali per la privacy delle risorse sono interne alle aziende, invece gli attacchi eserni rappresentano un rischio minore per la privacy. Gli attacchi esterni si possono riperquotere sugli altri attributi di QoS (come, scalabilià, performance o disponibilità). Infatti i modelli presenti oggio girno partano da elementi di analisi degli altri attributi per determinare possibili comportamenti anomali.
Altri problemi nascono con l'utilizzo di cloud ibride, e la loro prevenzione viene fatta seguendo eventuali politiche di definizione di un SLA appropriato.

Gli attributi di QoS presentati e discussi nella tesi sono difficilmente trattabili in maniera univoca, siccome sono posti in relazioni molte stretta l'una con l'atra. Perciò un qualsiasi approccio preventivo per la QoS deve essere eseguito tenendo conto dell'interazione degli attibuti e su come l'effetto di uno si riperquote sull'altra. Questo effetto non si utilizza solo per effettuare il recupero del sistema ma ancora più fondamentale è come esso può essere utilizzato per determinare un eventuale anomalia.

Ringraziamenti

Io sottoscritto Ludovico Goooobba mi laureo!

E queste ultime righe che scrivo come studente ho deciso di dedicarle a voi, parenti, amici e professori che siete stati il mio sostegno morale per la durata dei miei studi, e sono certo che lo sarete anche in futuro. Io, "un povero pazzo", che senza il vostro sostegno non sarei andato da nessuna parte. Grazie!
Ho tante parole e tanti pensiri da esprimere, e come al solito, mi trovo agli sgoccioli. Ora cerco di fare una piccola sintesi di ciò che ho in testa, spero di non trascurare nessuno.

Egregio **Prof. Carlo Ferrari** la ringrazio per il suo sostegno. Inoltre Le volevo dire che Lei è una persona gentile e generosa, ed e molto gradevole comunicare con Lei.

Mamma e Pappà (El & Lem), vi ringrazio per l'affetto che mi date, per il duro lavoro e i sacrifici che avete fatto per portarmi fino a questo giorno, non avete la più pallida idea di quanto vi stimo. *Ju a shperblesha në gzime të mëtejshme.* **Nonna Tere**, la mia voce, il mio sostegno, sei fantasica. Non so quanto sarei riuscito a sopravvivere senza il tuo strudel alle mele. **Zio Zef**, ti ringrazio per le lunghe conversazioni e sono molto orgoglioso di te.

Xhon Çoba, quest'ultima settimana sei stato il mio supereroe. Mi piace quando ci prendiamo a sberle, anche se fa male, però so che alla fine torniamo ad andare bene come prima(O quasi! Ti spacco!). Sono certo che in un futuro molto vicino

troverai quello che mi hai chiesto di non scrivere in questo ringraziamento.

Questo spazio le dedico a quelli che mi hanno fatto ubriacare, incazzare, urlare, battere il cuore, giocare a calcio e apprezzare la mia città natale.

Grazie **Kristi** per un'infinità di ragioni, mi ci vorrebbe un'altra tesi per spiegarle tutte. Sei stata la mia musa, motivo di conforto e sconforto, confusione e ragione, un pò di tutto insomma. Sei tra le persone a me più care in questo mondo. Grazie!

Allora, **Jozef e Sander** siete due pezzi di mmmmm. Mi sono rotto di giocare a calcetto contro di voi e perdere sempre! Dobbiamo sistemare i team sennò io non gioco più. Ma siccome devo ringraziarvi e non maledirvi, il primo lo ringrazio per le strepitose serate all'insegna dell'alcohol, invece l'altro grazie per il caffe mattutino e i consigli. **Rozi** ti ringrazio perché mi fai sentire molto a mio agio quando parliamo insieme, non mi trovo così bene con tutti. **Marsi** grazie perché senza di te per me le discoteche non esisterebbero. **Zef Haxhia** se la parola amico potesse prendere una forma umana si incarnerebbe dentro di te. **Damira e Henri** grazie per la nuova macchina. **Lucjan** perché prima di conoscerti pensavo che il calcio non esistesse. **Kole Kurti** altro pazzo che naviga in un mare senza vento. Le sessioni di True Blood insieme a te sono state fantastiche, mi dispiace veramente tanto che tu sia scappato, però sono certo che tornerai a trovarci.

Un ringraziamento particolare va a **Universiteti i Shkodrës Luigj Gurakuqi**, sponsor generale per il mio cazzeggio. Scherzi a parte, mi ha aiutato a mettermi in riga.
Ringrazio il Magnifico Rettore **Prof. dr. Artan HAXHI** per avre creduto in me. Inoltre ringazio tutto i miei colleghi: **Fatmir, Virtyt, Erikliza,** $Zana^3$**(le tre Zana), Suada, Saida, Sander, Mimi, Genci, Arta.** Un ringraziamento particolare va a **Gezim** per le gradevoli pause caffé. **Ana Karenina Berisha**per la sua unicità e carisma. **Lidia** per la sua (20:12 qua ho stallato per un attimo!!!)(20:49 sono ancora in stallo)(23:37 porca miseria) purezza(**04:07 AM del giorno dopo...**).

Erard perchè sei furbetto. **Diana** perché sei gentile.

Ringrazio **Florian Bjnaku, Gjovalin Deda e Mark Rupa** perché prima di conoscere voi pensavo che alla mia "venerevole" età non avrei più trovato delle amicizie sincere e solide. Grazie a voi ho capito che ero in torto. Poi c'è da aggiungere che un pò siete degli stronzetti!

Ringrazio il Dipartimento di Informatica a Scutari. In particolare **Genc & Sidita**. Anche **Andi** che è un grandissimo amico. **prof. dr. Fatos Kopliku** per la sua sincerità.

Ora ringrazio tutti i miei amici Patavini. Ho intenzione di citarvi seguendo una cronoliga su quando vi ho conusciuti, partendo dal mio primo anno.

Devo cominciar con **Pierangelo Marchesini**, primo stronzo che ho conosciuto a Padova. Sei il re dei cazzoni. Ogni volta che sto con te mi stra diverto, la tua risata è insormontabile. Poi insieme sfottiamo il Mala. **Stefano Organo e Daniele Sartori**, maestri della baruffa. Siete stati fondamentali per deviarmi verso una vita di cazzeggio. **Gigi Tibaldo** sei un re, il genio tra i ginii. **Icio Conicio** il pazzo tra i pazzi. **Orcuz** altro pazzo ma che si è guadagnato tutta la mia stima. **Davide Bubba Brunelli** sei il mio compagno inseparabile. Non dimenticherò mai le ore spese insieme a vedere anime e a leggere manga. C'è da citare War Craft, Lord of the Rings, SimCity2000... L'anno insieme in appartamento è stato bellissimo. **Marco Gasparato** lo so che pretendi il mio amore ma io non te lo posso concedere se prima non ci sposiamo su Facebook. Tu sei forse una delle persone con la mente più creativa per quanto riguarda le stronazzate. Indimenticabile il nostro secondo anno all'università a cantare tutte le sere sul balcone. **Mala** sei come un pezzo di pa(e)ne, sei un bravo ragazzo, non bevi, non bestemmi e sporattutto sei bello(secondo tua sorella). Mi diverto tantissimo con te e devo ammettere che quando ti incazzi diventi veramente bello. **Adriano Zabot** sei un altro influsso

negativo nella mia vita. Non ti rendi conto di quanto mi hai deviato. Grazie a te ho imparate a rockeggiare come in Tenacious D, e la bestemmia. Indimenticabili tutti gli istanti di cazzeggio spesi insieme. **Marco Bettiol** sei un punto di riferimento per me, sei bravo in tutto ciò che fai, ti stimo tantissimo. Io non so veramente come fai a fare tutto ciò che fai. **Cum Gian Franco** altro scienziato pazzo. Per furtuna che ti sei convertito alla tecnologia e ora mentre lavoro possiamo scambiargi messaggini porchi su whatsapp. **Scaglia e Rosicki** perchè siete due grandi persone. **Ontuz** la mia matricolina preferita. Non dimenticare mai di usare il tool secchio. Grazie **Photoshop** che mi hai concesso di immortalare momenti indimenticabili con Ontuz. **Imprex** che sei un extreme gamer, impossibile batterti in qualsiasi giochetto, sei un maestro a maneggiare il tuo "joystick". **Enrica D'Elia** grazie per i "pup" su Facebook. **Marco Google** sei una pistola. Sei stra in gamba vecchio e sono certo che un giorno riuscirai a spaccare e a raggiungere i tuoi obbiettivi. **Thomas Aragosta** diventerai un medico molto pazzo, i tuoi pazienti saranno vitteme dei tuoi esperimenti assurdi. Cmq mi piaci perché sei sempre allegro ed è stra piacevole scambiare idee con te. **Franco Bortottolenco** erede sofisticato di Cum, un mostro accademico e indubbiamente un buon amico. **Boatto** perché sei il più simpatico pazzo che conosco. **Simone Tescari** che prima mi odiava poi ha cominciato ad amarmi. Grazie perchè ci hai rovinato la vacanza al lago di Garda. Sei assurdo in tutto ciò che fai ma possidi un carisma unico. Prima di conoscerti pensavo che tu fossi gay. **Elena Cereser** migliore compagna femmina di bevute che ho mai avuto, e sicuramente batti anche tanti maschi ubriaconi. Stra bello stare allo Zanellato a bere con te e fare il giochetto stupido che ci siamo inventati con lo stecchino! **Wooduz** nonostante tu sia pazzo so che quando hai un obbiettivo in testa, anche se assurdo, tu ce la fai a raggiungerlo. Sei alcolizzato e mi piaci per questo. **Diona Gjermeni** sei una persona stra buona e brava, sono certo che diventerai un medico eccezzionale. ringrazio anche **Hajar** perché è semplicemente mitica.

Vogio inoltre ringraziare **Internet, l'alcohol, le passere e l'ERASMUS** perché mi hanno fatto perdere un infinità di tempo, che avrei potuto impiegare a studiare. Inoltre grazie agli orrori ortografici che mi fanno fare una brutta figura

ma non farò mai in tempo a sistemarli tutti.

Bibliografia

[1] Luis M. Vaquero, Luis Rodero-Merino , Juan Caceres, Maik Lindner, *A Break in the Clouds: Towards a Cloud Definition*,Gennaio 2009.

[2] Peter Mell and Timothy Grance, *The NIST Definition of Cloud Computing*, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2009.

[3] Lizhe Wang, Jie Tao, Marcel Kunze, *Scientific Cloud Computing: Early Definition and Experience*, 2008.

[4] The RESERVOIR Project, `http://www.reservoir-fp7.eu/`, 5 marzo 2009.

[5] Amazon Elastic Compute Cloud(EC2), `http://aws.amazon.com/ec2/`, agosto 2011.

[6] OpenNEbula Project, `http://www.opennebula.org/`, giugno 2011.

[7] Nimbus Project, `http://www.nimbusproject.org/`, settembre 2012.

[8] Salvatore Distefano, Antonio Puliafito, Massimiliano Rak, Salvatore Venticinque, *QoS Management in Cloud@Home Infrastructures*, 2011.

[9] The Eucalyptus Cloud, `http://www.eucalyptus.com/eucalyptus-cloud/iaas`, 2012.

[10] The Hadoop Distributed File System, `http://www.aosabook.org/en/hdfs.html`, 2012.

[11] Amazon's Simple Storage Service (S3), `http://aws.amazon.com/s3/`, 2012.

[12] Trieu C. Chieu, Ajay Mohindra, Alexei A. Karve, *Scalability and Performance of Web Applications in a Compute Cloud*, 2011.

[13] Jae Yoo Lee, Soo Dong Kim, *Software Approaches to Assuring High Scalability in Cloud Computing*, 2010.

[14] Ruiqing Chi, Zhuzhong Qian and Sanglu Lu, *A Heuristic Approach for Scalability of Multi-tiers Web Application in Clouds*, 2011.

[15] Hai-Mei Xu, Yan-Jun Shi, Yu-Lin Liu, Fu-Bing Gao, Tao Wan, *Integration of Cloud Computing and P2P: A Future Storage Infrastructure*, 2012.

[16] J. Gozdechi, A. Jajszczyk, R. Stankiewicz, *Quality of Service Terminology in IP Networks*, 2003.

[17] Eeraj Jan Qaisar, *Introduction to Cloud Computing for Developers*, 2012.

[18] William Voorsluys, James Broberg, Rajkumar Buyya, *Introduction to Cloud Computing*, febbraio 2011.

[19] J. Schad, J. Dittrich, and J.A. Quiane-Ruiz, *Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance*, 2010.

[20] Jin Shao, Qianxiang Wang,*A Performance Guarantee Approach for Cloud Applications Based on Monitoring*, 2011.

[21] Qiang Duan, *Modeling and Performance Analysis on Network Virtualization for Composite Network–Cloud Service Provisioning*, 2011.

[22] Jim (Zhanwen) Li, John Chinneck, Murray Woodside, Marin Litoiu, Gabriel Iszlai, *Performance Model Driven QoS Guarantees and Optimization in Clouds*, 2009.

[23] Greg Franks, Tariq Al-Omari, Murray Woodside, Olivia Das, Salem Derisavi *Enhanced Modeling and Solution of Layered Queueing Networks*, 2009.

[24] Andres Quiroz, Hyunjoo Kim, Manish Parashar, Nathan Gnanasambandam, Naveen Sharma, *Towards Autonomic Workload Provisioning for Enterprise Grids and Clouds*, 2009.

[25] Rodrigo N. Calheiros, Rajiv Ranjany, and Rajkumar Buyya, *Virtual Machine Provisioning Based on Analytical Performance and QoS in Cloud Computing Environments*, 2011.

[26] Young Choon Lee, Albert Y. Zomaya, *Rescheduling for reliable job completion with the support of clouds*, 2010.

[27] Michael Armbrust, Armando Fox ,Rean Griffith,Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia, *Above the Clouds: A Berkeley View of Cloud Computing*, febbraio 2009.

[28] I. Sommervile, *Software Engineering 8th*, 2006.

[29] SLA@SOI, `http://research.xlab.si/index.php?option=com_content&task=view&id=151&Itemid=161`, 2012

[30] A. Cuomo, M. Rak, U. Villano, *Chase: an autonomic service engine for cloud environments,* 2011

[31] R. Aversa, B. Di Martino, N. Mazzocca, S. Venticinque, *Magda: A mobile agent based grid architecture*, 2006.

[32] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, *Web services agreement specification (ws-agreement)*, 2004.

[33] Davide Lorenzoli, George Spanoudakis, *Predicting Software Service Availability: Towards a Runtime Monitoring Approach*, 2011.

[34] Raúl Gracia-Tinedo, Marc Sánchez-Artigas, Pedro García-López, *F2BOX: Cloudifying F2F Storage Systems with High Availability Correlation*, 2012.

[35] Keith Jeffery, Burkhard Neidecker-Lutz, *THE FUTURE OF CLOUD COMPUTING*, 2010.

[36] Zhengping Wu, Nailu Chu, Peng Su, *Improving Cloud Service Reliability - A System Accounting Approach*, 2012.

[37] D. Pao, W. Lin, B. Liu, *A memory-efficient pipelined implementation of the aho-corasick string-matching algorithm*, 2010.

[38] G. I. Webb, J. R. Boughton, Z. Wang, *Not So Naive Bayes: Aggregating One-Dependence Estimators Machine Learning*, 2005.

[39] Qingqing Feng, Jizhong Han, Yun Gao,Dan Meng, *Magicube: High Reliability and Low Redundancy Storage Architecture for Cloud Computing*, 2012.

[40] Akhil Behl, *Emerging Security Challenges in Cloud Computing*, agosto 2011.

[41] Pankesh Patel, Ajith Ranabahu, Amit Sheth, *Service Level Agreement in Cloud Computing*

[42] K. Bernsmed, M. G. Jaatun, and A. Undheim, *Security in Service Level Agreements for Cloud Computing* , 2011.

[43] Guo Zhien, Day Yiqi, *Security SLAs for IMS-based Cloud Services*, 2012.

[44] J. Skene, D. Lamanna, W. Emmerich, *Precise Service Level Agreements*, 2004.

[45] Ali Hammami, Noëmie Simoni, Rasha Salman, *Ubiquity and QoS for Cloud Security*, 2011.

[46] Karin Bernsmed, Martin Gilje Jaatun, Per Hakon Meland, Astrid Undheim, *Security SLAs for Federated Cloud Services*, 2011.

[47] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, *The Google File System*, 2003.