POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Analisi Simulativa di Algoritmi Class-Based per Load Balancing di macchine virtuali in ambiente Cloud

Relatore: Prof.re MARCO GRIBAUDO

Tesi di Laurea Magistrale di:

FEDERICO PRELI Matricola 766615

Anno Accademico 2012 - 2013

RINGRAZIAMENTI

Un ringraziamento a tutte le persone, amici e parenti, che mi hanno aiutato, incoraggiato, sostenuto e che mi sono state vicine durante il percorso universitario.
Un grazie anche al professor Marco Gribaudo che con la sua pazienza, la sua disponibilità e la sua sapienza, mi ha aiutato e consigliato nella realizzazione di questo progetto di tesi. Ma soprattutto grazie alla persona che, con il suo silenzioso incitamento, ha fatto si che io potessi arrivare fino a qui, grazie Papà.

ABSTRACT

Oggi le aziende dipendono dall'IT per favorire l'innovazione e accelerare il passo rispetto alla concorrenza. Il ricorso sempre più frequente al cloud computing permette alle aziende di offrire nuovi servizi, penetrare in nuovi mercati, avvicinarsi ai clienti e aumentare la produttività di dipendenti sempre più esigenti e "mobili". Nelle organizzazioni di maggior successo, le tecnologie cloud fungono da catalizzatore per l'implementazione di nuovi sistemi e processi che consentono di ottimizzare, non solo le metriche relative all'operatività, ma anche i risultati aziendali [1]. Uno degli aspetti più importanti del Cloud è la suddivisione delle risorse per le richieste in ingresso al sistema. In questo trattato di tesi abbiamo deciso di effettuare un'analisi simulativa e critica degli algoritmi class-based per il load balancing su macchine virtuali in ambiente Cloud. L'implementazione e la simulazione di tali algoritmi sono stati realizzati grazie ad uno strumento molto utilizzato in ambito accademico, ovvero Omnet++. I risultati ottenuti, come vedremo, rispecchiano l'obiettivo principale del nostro elaborato, ovvero il miglioramento, in termini prestazionali, dei meccanismi esistenti attraverso la loro sostituzione con nuovi algoritmi di nostra creazione. Per avere una visione più chiara dell'ossatura del nostro elaborato, esponiamo ora un breve excursus degli argomenti trattati.
Questo progetto di tesi si compone di una prima parte introduttiva con il fine di descrivere il Cloud Computing, le sue caratteristiche e le maggiori soluzioni presenti ad oggi sul mercato. Il primo capitolo si conclude con una descrizione del problema del Load Balancing e di tutti i suoi aspetti chiave. Nei capitoli due e tre, poi, viene realizzato un breve accenno alla simulazione ad eventi discreti, analizzando e descrivendo inoltre l'ambiente simulativo Omnet++. Il capitolo 4, comprende la descrizione del modello utilizzato nel progetto, attraverso un analisi di tutti i suoi componenti e delle loro funzionalità principali. La parte conclusiva dell'elaborato (capitoli 5 e 6) è dedicata all'analisi simulativa e prestazionale dei vari algoritmi trattati e all'esposizione di alcune tematiche di sviluppo futuro.

PAROLE CHIAVE: Analisi simulativa, algoritmo, Load Balancing, Cloud Computing, macchina virtuale, Omnet++

Indice

Elenco delle figure

Elenco delle tabelle

1 INTRODUZIONE

Negli ultimi anni è salita alla ribalta mediatica una tecnologia che ha cambiato radicalmente le nostre abitudini di internauti. Questa tecnologia, come ampiamente descritto in "Above the Clouds: A Berkeley View of Cloud Computing" (2009) [2], nasce da un'idea concepita diversi anni fa, ma che recentemente ha visto aumentare la sua notorietà e il suo utilizzo in maniera esponenziale. Stiamo parlando del Cloud Computing (letteralmente Nuvola Informatica). Ad oggi, tantissime persone, anche senza particolari capacità informatiche, ne fanno uso quotidianamente o almeno in un occasione hanno avuto modo di imbattersi in questa tecnologia. Il numero elevatissimo di pubblicità, soprattutto in rete, ci fa capire quanto questo fenomeno stia crescendo di portata, ed è facile auspicare come, in poco tempo, esso possa diventare un vero e proprio "oggetto" di uso comune.
Quando si parla di Nuvola non si parla solamente di Internet, ma si parla di una tipologia di servizio vero e proprio, i cui punti di forza stanno sia nella facilità di accesso e utilizzo, sia nel concetto di "Pay-to-use" che vedremo in seguito. In questo capitolo, partendo da una definizione generale, andremo ad approfondire il tema del Cloud Computing, analizzandone pregi e difetti e presentando, nella prima parte, una breve carrellata dei possibili approcci adottabili dalle aziende presenti oggi sul mercato. Nella seconda parte invece, introdurremmo la caratteristica del Cloud su cui si basa questo elaborato, ovvero il Load Balancing, utilizzando come strumento di confronto e paragone il modello EC2 di Amazon. Infine, nella terza ed ultima parte, andremo a definire i problemi etici ed informatici introdotti dall'avvento della Nuvola.

1.1 IL CLOUD COMPUTING

1.1.1 DEFINIZIONE E STORIA

In informatica, con il termine inglese Cloud Computing (in italiano nuvola informatica), si indica un insieme di tecnologie che permettono, tipicamente sotto forma di un servizio offerto da un provider al cliente, di memorizzare/archiviare e/o elaborare dati (tramite CPU o software). Tutto questo grazie all'utilizzo di risorse hardware/software distribuite e virtualizzate in Rete.
E' noto che, utilizzando varie tipologie di unità di elaborazione (CPU), memorie di massa fisse o mobili come ram, dischi rigidi interni o esterni, Cd/DVD, chiavi USB, eccetera, un computer sia in grado di elaborare, archiviare e recuperare programmi e dati. Nel caso di computer collegati in rete locale (LAN) o geografica (WAN) questa possibilità può essere estesa ad altri computer e dispositivi remoti dislocati sulla rete stessa. Sfruttando la tecnologia del Cloud Computing, così, gli utenti collegati ad un cloud provider (letteralmente un fornitore di servizio di cloud) possono svolgere tutte queste mansioni, anche tramite un semplice accesso ad un internet browser. Possono, ad esempio, utilizzare software remoti non direttamente installati sul proprio computer, salvare dati su memorie di massa on-line predisposte dal provider stesso (sfruttando sia reti via cavo che senza fili) e cosi via.
Prima di addentrarci nel mondo Cloud con tutte le sue caratteristiche, è necessario e doveroso presentare un excursus storico dei passi che hanno portato questa tecnologia alla ribalta negli ultimi anni.
Agli albori dell'era computazionale, nei CED erano collocati enormi mainframe e, chi avesse voluto farne uso, avrebbe dovuto prenotarne l'utilizzo, recarsi al CED con uno scatolone pieno di schede perforate (sulle quali era registrato il programma da far eseguire) ed attendere i risultati. Quando il costo dei componenti elettronici iniziò a diminuire, dai grossi ed esosi mainframe, si passò ai MiniComputer ed anche le medie aziende e le università iniziarono a potersi permettere un proprio calcolatore. C'è da sottolineare che il paradigma di comunicazione utilizzato in quegli anni (siamo negli anni '70-'80) non era client/server, perché i terminali, dai quali gli utenti avevano accesso al mainframe/minicomputer, erano solo un mero rimpiazzo delle schede perforate e delle stampanti ad aghi: un terminale non faceva altro che fornire una tastiera ed un monitor in una stanza

che non fosse quella in cui erano ubicati i rack del mainframe. Avvicinandosi ai nostri giorni, negli anni '80 affiorò il termine personal computer e nelle aziende iniziò a sorgere il problema di come poter condividere facilmente risorse tra i pc in dotazione ai dipendenti: sale in auge l'architettura Client/server e si inizia a parlare di File Server, Server di Stampa, Server di autenticazione, etc. Con l'arrivo delle connessioni dialup, le aziende poterono iniziare ad interconnettere le proprie filiali in modo semplice e le compagnie telefoniche iniziarono ad avere, a listino, servizi a valore aggiunto per le aziende e per gli utenti finali. La prima azienda a configurarsi come ISP fu Prodigy, che forniva un accesso dialup e diversi servizi, come forum di discussione, email, servizi bancari ed una piattaforma di ecommerce. L'avvento del Web e l'aumentare della banda disponibile per i collegamenti portò alla nascita dei primi Application Service Provider (ASP), ovvero delle prime aziende che fornivano applicazioni accessibili tramite accesso remoto. Purtroppo i tempi non erano ancora maturi e le soluzioni proposte non ebbero il successo sperato a causa dei costi delle licenze e per le stesse ragioni che oggi ancora ci spingono a guardare con diffidenza il cloud computing (ad es. proprietà dei dati, qualità del servizio, etc). Accantonata l'idea di fare uso massivo dei servizi sviluppati dagli ASP, Connectix prima, VMWare poi, presentarono alle aziende i loro prodotti per la virtualizzazione. VMWare[1] Workstation ed ESX Server fecero il loro esordio a cavallo degli anni 2000 e furono accolti positivamente dalle aziende, che videro nella virtualizzazione la possibilità concreta di ottimizzare l'infrastruttura IT. Proprio la virtualizzazione è uno dei pilastri su cui poggiano i servizi cloud attualmente erogati dai provider mondiali. Agli inizi del terzo millennio, poi, nasce l'idea di una nuova architettura, in cui l'infrastruttura software viene vista come un insieme di servizi organizzati per concorrere allo stesso obiettivo; l'architettura in oggetto viene chiamata Service Oriented Architecture (SOA) che nel corso del passato decennio viene completamente definita. Il paradigma SOA è stato ed è tutt'ora adottato da molti vendor, per gli indubbi vantaggi che l'architettura porta con se. La fusione tra le tecnologie di virtualizzazione ed il paradigma SOA ha permesso lo sviluppo di applicativi altamente portabili e ad architettura distribuita, oltre alla possibilità di interconnettere ed utilizzare facilmente software eterogenei ed eventualmente gestiti da altre aziende. Le potenzialità delle infrastrutture cloud, quindi,

sono possibili solo grazie all'applicazione di quanto descritto; questo vuol dire che le infrastrutture cloud non fanno altro che utilizzare tecnologie mature e sviluppate nel corso dell'ultimo decennio. Venendo agli ultimi anni, Amazon ha svolto un ruolo chiave nello sviluppo del cloud computing attraverso la modernizzazione dei data center, consentendo di sfruttarne tutte le potenzialità e lanciando Amazon Web Service (AWS) su una base di utility computing già dal 2006. Nel 2008, poi, si sono sviluppate altre soluzioni concorrenti, come Eucalyptus (il primo open-source), OpenNebula, software open-source per l'implementazione di cloud privati e ibridi, fino ad arrivare ai giorni nostri con la nascita di tantissime soluzioni di cui ci occuperemo tra poco.

1.1.2 TIPOLOGIA DI SERVIZI CLOUD

Dopo questa breve introduzione storica, passiamo ora a vedere quali tipologie di servizi, già descritte da C. Hfer and G. Karagiannis (2011) [3], può offrire il Cloud Computing. Ad ognuna di esse, allegheremo degli esempi di servizi ad oggi disponibili in commercio, così da avere una panoramica generale del mercato Cloud attualmente presente.
Ad oggi possiamo distinguere tre categorie fondamentali di servizio:

1. SaaS (Software as a Service)
   Questa modalità di cloud consente agli utenti di utilizzare software direttamente dai loro computer (o devices) attraverso il browser, senza la necessità di dover gestire l'infrastruttura cloud e la piattaforma su cui l'applicazione è in esecuzione. Questo elimina l'onere di installare ed eseguire il programma sul computer dell'utente, al quale non rimane altro che sfruttare al massimo le potenzialità del SW preso in "prestito".
   Generalmente la modalità di pagamento delle applicazioni SaaS è un canone mensile o annuale, ma è facile trovare in rete soluzioni gratuite con un ottimo compromesso qualità/prezzo. Per capire la portata di questo fenomeno, basta sapere che ad oggi, ci sono più di cento milioni dispositivi di elaborazione collegati a Internet e molti di loro utilizzano servizi di cloud computing quotidianamente. Secondo l'anticipazione di IDC [4], a livello economico, il mercato di SaaS (Software As A Service) ha raggiunto 13,1 miliardi dollari di ricavi al 2009 e aumenterà fino

a 40,5 miliardi dollari entro il 2014 ad un tasso di crescita annuale composto (CAGR) del 25,3%.
Grazie alla presenza di un mercato così florido, molte colossi informatici hanno deciso di sviluppare la propria soluzione SaaS tra le quali ricordiamo:

GOOGLE
Le applicazioni SaaS offerte da Google [5], dette Google Apps, sono le più sfruttate dagli utenti nel web e permettono di creare un ufficio virtuale vero e proprio, completamente accessibile dal web. I principali servizi offerti sono:
Google Mail: una casella di posta elettronica gratuita, che offre ai singoli utenti 7 GB (alle aziende 25 GB) per l'archiviazione di dati, garantiti da un filtro antispam che tiene lontana dal tuo account la posta indesiderata.
Google Docs: servizio di storage online di documenti.
Google Plus: il nuovo social network sviluppato da Google
Youtube: servizio online che permette di caricare e vedere video nel web in maniera completamente gratuita
Google Drive: la naturale evoluzione di Google Docs, che consente di archiviare files in rete.
Google Sites: è un modo semplice di creare pagine web senza ricorrere all'uso dell'HTML o della codifica. E' compatibile con più sistemi operativi come Windows Xp, Vista, Mac e Linux. Gli amministratori del sito, possono gestire le autorizzazioni di condivisione dei siti a livello aziendale, mentre gli autori possono condividere e revocare l'accesso ai file in qualsiasi momento.

APPLE ICLOUD
ICloud è il servizio offerto dalla casa di Cupertino che fa della semplicità di utilizzo il principale punto di forza. Di contro però rimane pur sempre un servizio focalizzato sui dispositivi e sulle applicazioni che ruotano intorno alla piattaforma iOs e Os X. La possibilità di utilizzarlo con devices Windows c'è, ma le sue virtù si evidenziano maggiormente con le applicazioni scaricate dall'App Store e utilizzate sui computer Mac, o su devices quali iPad e iPhone.

MICROSOFT SKYDRIVE
SkyDrive permette di utilizzare cartelle remote del pc per archiviare e sincronizzare file (la condivisione pubblica dei documen-

ti, come nel caso di Google Drive, richiede invece l'obbligo del browser Web), supporta iPhone e iPad e si prepara ad operare con Windows 8. Presto sarà inoltre anche integrato con il servizio di sincronizzazione Live MeshSkyDrive, tramite cui tutti i file memorizzati via cloud saranno automaticamente mantenuti aggiornati fra pc, Mac e il sito SkyDrive.com.

2. HaaS (Hardware As A Service)
   In questa tipologia di servizio, l'Hardware viene reso disponibile in modalità Cloud a fronte di un contratto di affitto o noleggio. Come descritto da George Reese in "Cloud Computing. Architettura, infrastrutture, applicazioni"[6] si può chiedere un server nel momento esatto in cui si ha bisogno e di li a poco, circa 10 minuti, esso è pronto all'uso. Quando non ci servirà più basterà semplicemente disattivarlo e il contratto cesserà di esistere. Dal punto di vista economico, anche in questo caso, vige la regola del "Pay-to-use", ovvero viene corrisposto un contributo che varia in base al tempo di utilizzo di tale bene o servizio, il che è considerato un grosso punto di forza dai consumatori. Le aziende, dal canto loro, trovano grande giovamento dall'utilizzo di servizi HaaS infatti se durante la realizzazione di un progetto si accorgono di aver sbagliato la pianificazione delle risorse HW(ottimisticamente o pessimisticamente), attraverso la Nuvola è possibile andare a correggere tale errore quasi in tempo reale, con dei costi minori rispetto al caso in cui l'HW venga gestito direttamente dall'azienda. Inoltre in caso di guasti ad un server, per esempio, la Nuvola permette di garantire un alto livello dìì la fault-tolerance alle aziende, sostituendo l'apperecchio guasto e ripristinando il servizio praticamente Real Time. In termini economici e ambientali, se la componente HW di un azienda fosse gestita autonomamente e internamente, i costi dovuti al consumo di energia elettrica e allo spazio occupato sarebbero maggiori rispetto ad una soluzione HaaS.

3. DaaS (Data As A Service)
   L'uso di Hardware virtuale sulla Nuvola non riguarda solamente i server, come visto precedentemente, ma anche veri e propri spazi virtuali o meglio dischi rigidi virtuali che vanno a sostituire i dischi rigidi fisici. Il Cloud Storage permette la memorizzazione dei dati direttamente sulla nuvola senza preoccuparsi,

ne dello spazio disponibile sul disco fisso del nostro pc, ne tantomeno dei meccanismi fisici di Backup e salvataggio. E' facile intuire che il punto di forza di questa tipologia di servizio sia la grande facilità di accesso ai dati. Ovunque ci troviamo infatti, è possibile accedere alle nostre informazioni semplicemente attraverso una connessione ad internet. Purtroppo però, ciò che spesso l'utente finale ignora è che il servizio DaaS non è stato progettato allo scopo di sostituire i dischi rigidi veri e propri, ma come mezzo di Backup dei dati, in quanto ad oggi non ancora in grado di garantire livelli prestazionali adeguati a tale scopo.

A queste tre tipologie di servizi, possono essere integrate altre due categorie da essi derivate che sono in costante crescita e vedono aumentare il loro numero di clienti in maniera esponenziale. Stiamo parlando dei servizi PaaS e IaaS.
Con il termine PaaS (Platform as a Service) ci riferiamo ad un servizio in cui i fornitori di cloud offrono una piattaforma di calcolo e/o un "pacchetto" di soluzioni. tra cui, il sistema operativo, il linguaggio di programmazione, l'ambiente di esecuzione, i database, i server web e così via. Gli sviluppatori di applicazioni, utenti di tale servizio, sono così in grado di gestire le proprie soluzioni software su una piattaforma Cloud, senza la preoccupaazione dovuta ai costi derivati dall'acquisto e dalla gestione dell'hardware e del software altresi necessari. Nella maggior parte delle offerte PaaS inoltre, il software gestisce automaticamente l'allocazione delle risorse necessarie a soddisfare la domanda dell'utente finale nel miglior modo possibile. Tra i servizi attualmente in commercio spiccano:

- Windows Azure: piattaforma cloud che offre servizi applicativi di storage, di infrastruttura e di networking, supportati dai server che operano all'interno della rete globale dei datacenter Microsoft. Tutto questo consente all'utente di attivare applicazioni sia in modalità cloud, sia localmente, abilitando numerose possibilità di sviluppo in un'ampia gamma di scenari che coinvolgono aziende, consumatori ed enti pubblici. Gli sviluppatori hanno la possibilità, inoltre, di selezionare un'ampia gamma di strumenti e tecnologie provenienti dal mondo open source o da altri software vendor e accedere ai servizi di Azure.

- Amazon Web Services: attivo dal 2006, Amazon Web Services (AWS), offre alle aziende di tutte le dimensioni una piattaforma di servizi Web basati su cloud computing. Con AWS è possibile richiedere potenza di calcolo, capacità di storage e tutta una serie di servizi che permettono di accedere ad una infrastrutture pensate per l'ambiente IT. AWS offre la flessibilità necessaria per scegliere la piattaforma di sviluppo o di programmazione che meglio si adatta al problema che si sta tentando di risolvere. Negli ultimi anni è diventato anche un'alternativa ai servizi di hosting tradizionali. AWS ad oggi è diventato il modo più economico per sviluppare applicazioni di una certa rilevanza ed è il servizio Cloud più utilizzato in ambito accademico. Nel proseguo della tesi ci sarà un importante approfondimento su un aspetto chiave di AWS ovvero Amazon EC2.

- Google App Engine : Essenzialmente è un servizio che permette di ospitare la propria Web Application all'interno dell'infrastruttura cloud di Google in modo semplice e relativamente economico. E' possibile accedere alla propria app sia tramite un sottodominio di "appspot.com", sia attraverso un proprio dominio che però deve essere configurato con le Google Apps. Google mette inoltre a disposizione due sandbox collaudate per sviluppare le proprie applicazioni: una in Java e l'altra in Python.

Passando all'analisi della seconda tipologia di servizio derivata, possiamo dire che l'IaaS (Infrastructure as a Service) comprende una serie di servizi che l'utente può sfruttare per creare una vera e propria infrastruttura virtuale. Questo vuol dire che il Cloud Provider mette a disposizione la possibilità di creare macchine virtuali, istanziare memorie per immagazzinare dati, configurare firewall e impostare bilanciatori di carico(Load Balancer) al fine di creare un'infrastruttura più conforme possibile alle esigenze del cliente. I fornitori di IaaS distribuiscono queste risorse su richiesta, grazie ai loro data center dislocati nel mondo. Anche in questo caso, similmente ai precedenti, il costo dei servizi IaaS si una basa sull'utility computing, ciò vuol dire che il costo riflette la quantità di risorse stanziate e consumate dai fornitori di tale servizio.
Di contro questa tecnologia richiede all'utente una conoscenza, seppur limitata dell'informatica, in quanto sono necessarie le capacità

di installare il sistema operativo virtuale (e i software ad esso correlati) e il loro costante aggiornamento al fine da godere delle massime potenzialità del servizio IaaS.
Esistono Cloud Provider infine, che offrono pacchetti comprendenti, sia il servizio IaaS, sia quello Paas. Un esempio di tale offerta è Ospit@ Virtuale [7] ovvero, l'offerta di Cloud Computing di Nuvola Italia indirizzata al mercato delle piccole e medie imprese italiane. Con Ospit@ Virtuale i server dei clienti vengono "virtualizzati" nei Data Center di Telecom Italia, utilizzando tecnologie in grado di fornire in tempo reale la capacità elaborativa richiesta dalle applicazioni, grazie alla condivisione, in sicurezza, delle risorse hardware rappresentate dalle piattaforme estremamente potenti ed affidabili. L'utente ha, inoltre, la possibilità di accedere ad una console web di gestione per consentire la modifica, in base alle esigenze, di tutti i principali parametri di funzionamento dei server virtuali.
Queste suddivisioni in tipologie sono descritte da una Figure tratta da "Cloud computing: state-of-the-art and research challenges"(2010)[10] che in maniera chiara e intuitiva riassume quanto detto fino ad ora.

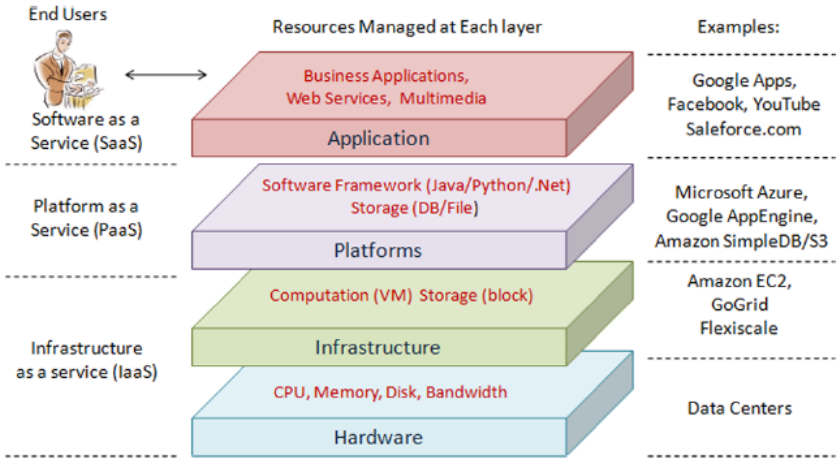

Figura 1: Tipologie Cloud Service

Dopo aver descritto le varie tipologie di servizi Cloud ad oggi disponibili, nel prossimo capitolo approfondiremo i pregi e i difetti che nascono dalla scelta di una struttura basata su Cloud rispetto alle

possibili strategie concorrenti ovvero l'outsouring e la gestione interna.

1.1.3 CLOUD COMPUTING, INFRASTRUTTURA INTERNA e OUTSOURCING

Come si è potuto intuire fino ad ora, una delle principali caratteristiche che ha reso cosi famoso il Cloud Computing è la trasparenza. Noi sappiamo che tale tecnologia rende disponibili all'utilizzatore le risorse come se fossero implementate da sistemi (server o periferiche personali) "standard", ma l'implementazione effettiva delle risorse non è definita in modo dettagliato; anzi l'idea è proprio che l'implementazione sia un insieme eterogeneo e distribuito –da qui il termine inglese the cloud– di risorse le cui caratteristiche non sono note all'utilizzatore. In sostanza l'utente finale non ha la necessità di preoccuparsi di come tali meccanismi vengano implementati, ma al contrario esso è portato ad utilizzarli in maniera del tutto naturale, caratteristica indispensabile per ogni buon Cloud Provider. Un esempio prettamente numerico di questo fenomeno e della sua portata ci viene fornito da uno studio condotto da Nextplora sui servizi Microsoft[8], il quale ci rivela come l' 88% di chi naviga su Internet usa almeno un servizio di cloud computing. Tuttavia lo fa in modo inconsapevole. A essere monitorato in ambito domestico è stato un campione di 1.000 cybernauti , dal quale è emerso inoltre che il 56% degli intervistati memorizza foto e filmati sul web, il 29% musica e compilation, mentre il 18% usa il cloud per archiviare documenti. Dal punto di vista informatico la notizia è positiva. Perché significa che il cloud computing assolve al suo compito principale, ovvero quello di operare attraverso App e software in modo trasparente, rendendo semplice la vita di chi lo usa.
Riassumendo quanto detto fino ad ora, con un numero così elevato di alternative possibili, a livello di servizi, per le aziende medio-piccole soprattutto, nasce la possibilità di non dover mai più acquistare un server o una licenza software ma semplicemente esse possono "affittarle" tramite provider. In altre parole tutte queste preoccupazioni tendono a sparire:
• Tutte le licenze software sono aggiornate? I sistemi SaaS mantengono aggiornati i loro SW costantemente e ad ogni nostro accesso a tali servizi troveremo sempre l'ultima release in commercio disponibile

• Cosa fare se un componente HW si rompe nella notte? Come detto i sistemi HaaS sono fault-tolerance quindi l'utente finale vedrà eliminato questo problema sul nascere infatti il server guasto verrà sotituito in maniera del tutto trasparente
• Come disfarsi dei vecchi server o aggiungere "potenza" al nostro sistema IT? Con i servizi HaaS basta scollegarsi ed il servzio termina oppure è sufficiente "affittare" nuovi server che questi in pochi minuti saranno pronti all'uso
• Come gestire gli ammortamenti degli asset IT? Poiché la politica di costo della Nuvola è "pay-to-use", non esistono investimenti in conto capitale da ammortizzare con un notevole beneficio sul bilancio annuale.

Tutti questi fattori dovrebbero portare ogni azienda ad adottare una soluzione basata su Cloud, rispetto ad una strategia tradizionale (ad infrastruttura interna) dove l'azienda stessa ha l'onere di acquistare fisicamente tutto ciò di cui ha bisogno e si vede costretta a spendere enormi quantità di denaro per mantenere aggiornata tale infrastruttura.
Esiste però un'altra tipologia di scelta possibile, ovvero l'Outsourcing a servizi gestiti da terzi. Quest'ultima categoria non si discosta molto dal concetto di Cloud, infatti anch'essa prevede la stipula di un contratto con un azienda esterna per la gestione dei servizi di cui si ha la necessità. La maggior parte di questi servizi si concentra sull'utilizzo di server di terzi a fronte di un canone mensile in modo da non doversi preoccupare dei problemi conseguenti derivati dalla gestione e manutenzione dell'architettura a infrastruttura interna. Ma allora quali sono le differenze che esistono tra l'outsourcing e la strategia basata su Cloud Computing? Per rispondere a tale quesito abbiamo deciso di approfondire alcuni aspetti fondamentali che caratterizzano queste due politiche:

1. COSTI INIZIALI: i costi iniziali del Cloud Computing sono bassisimi(spesso nulli) a fronte di investimenti sostanziali nella strategia basata su outsourcing
2. COSTI DI GESTIONE E MANTENIMENTO: i costi di gestione delle due soluzioni sono molto diversi, ancora una volta a favore della Nuvola
3. TEMPO DI DEPLOYMENT: il tempo di Deployment, ovvero di realizzazione e messa in opera del servizio è minore nel caso

di Cloud Computing in quanto, per esempio, bastano pochi minuti per accedere a nuovi server e integrarli alla rete contro i giorni necessari ad una strategia outsourcing based

4. COMPETENZE NECESSARIE: le competenze interne richieste risultano maggiori con una strategia Cloud, dove sono necessarie capacità di configurazione delle macchine virtuali richieste, a differenza dell'outsourcing in cui tutto è gestito da terzi

5. AFFIDABILITA': ad oggi, il livello di affidabilità del servizio Cloud non è ai livelli della strategia outsourcing, ma gli sviluppatori di Cloud Provider si stanno impegnando al fine di colmare questo gap.

Dopo questa analisi, basata sul confronte delle tre strategie esistenti per la politica IT di un'azienda, dovrebbe essere del tutto chiaro che, la creazione di un infrastruttura interna da zero non ha più molto senso. Le uniche aziende che percorrono questa strada sono quelle che in passato hanno effettuato investimenti significativi sull'infrastruttura o che per sicurezza dei dati non possono affidarli a terzi. Il resto delle realtà esistenti sono portate, a loro convenienza, ad optare per una strategia basata su outsourcing o Cloud Computing, il che ne favorisce lo sviluppo e la conseguente crescita commerciale.

1.1.4 ARCHITETTURA CLOUD

Come abbiamo visto, le caratteristiche che portano un azienda a scegliere una tecnologia come il Cloud Computing sono diverse, ma tutte dovute ad un attento studio, in fase di progettazione, di quella che è considerata l'architettura alla base del Cloud Computing [2]. Prima di visualizzarla e analizzarla nel dettaglio però, è necessario chiarire quali siano i ruoli possibili nei servizi offerti dal Cloud Computing.
Da una parte c'è il fornitore di servizi (Cloud Provider) che offre servizi generalmente attraverso un modello "pay-per-use", dall'altra il cliente che a sua volta si suddivide in due sottocategorie:

- Cliente Amministratore, il quale sceglie e configura i servizi offerti dal fornitore, generalmente offrendo un valore aggiunto al pacchetto come ad esempio applicazioni software;

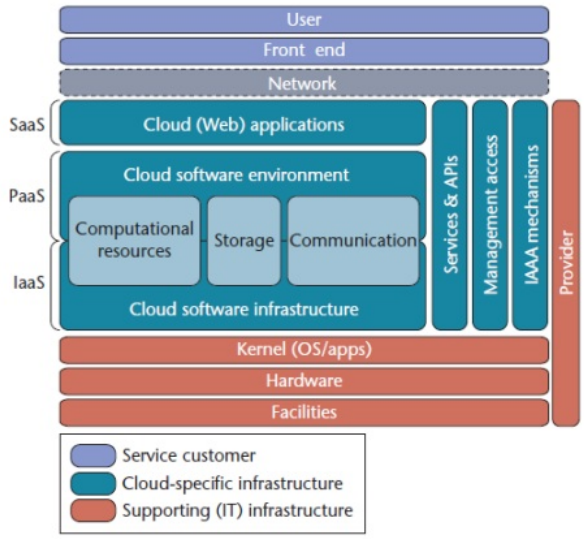

Figura 2: Architettura Cloud

- Cliente Finale, il quale utilizza i servizi opportunamente configurati dal cliente amministratore o direttamente dal Cloud Provider

In determinati casi d'uso, queste due sottocategorie possono coincidere, ad esempio un cliente che utilizza un servizio di Cloud Storage per effettuare il backup dei propri dati, è sia utente finale in quanto utilizzatore del servizio, sia cliente Amministratore in quanto configuratore di tale servizio.
Dopo questa piccola precisazione sui ruoli possibili nei servizi cloud, possiamo tornare all'analisi dell'architettura vera e propria che permette ai Cloud Provider di offrire tutti i servizi sopra elencati. Come possiamo immaginare essa è composta da uno o più server reali, generalmente integrati in un'architettura ad alta affidabilità e fisicamente collocati presso il data center del Cloud Provider, il quale espone le interfacce al fine di elencare e gestire i propri servizi. Il cliente amministratore, attraverso tali interfacce, ha la possibilità di selezionare il servizio richiesto e amministrarlo (configurazione, attivazione, disattivazione) a seconda delle esigenze del cliente finale, il quale potrà così godere di tale servizio a pieno regime.

Entrando maggiormente nel dettaglio tecnico, la Figure 2, mostra una possibile rappresentazione di architettura Cloud suddivisa in componenti logici.
Come si può vedere esistono 2 componenti principali: Front-End e Back-End.
Il Front-end è ciò che l'utente vede ed è ciò con cui l'utente interagisce, è installato sul dispositivo fisico che si utilizza per accedere ai servizi di Cloud e consiste nell'applicazione necessaria ad accedervi.
Il Back-end invece è il nucleo del sistema vero e proprio ed è collocato nelle sedi opportune dell'azienda fornitrice del servizio. Esso è composto da tutte le risorse e le unità di computazione necessarie affinché il servizio garantito, in termini di QoS, funzioni nel modo corretto e prestabilito in fase di stipula del contratto (SLA). E' importante ricordare inoltre, che in questa architettura è presente uno strumento di Monitoring che permette di monitorare lo stato del sistema e le sue effettive prestazioni. Quest'ultimo aspetto ci porta ad introdurre un concetto fondamentale del Cloud Computing e che rappresenta il cuore del nostro trattato di tesi ovvero il LOAD BALANCING.

1.2 LOAD BALANCING

In questo paragrafo andremo a spiegare la funzionalità di bilanciamento del carico in ambiente Cloud ovvero il Load Balancing. Nel primo paragrafo definiremo tale meccanismo avvalorando la nostra tesi con una breve descrizione dei pregi che introduce nel sistema. Nella seconda parte invece, analizzeremo nel dettaglio come Amazon, più precisamente AWS, implementa tale caratteristica attraverso lo studio di Amazon EC2.

1.2.1 DEFINIZIONE

Prima di dare la definizione di Load balacing, è necessario introdurre un concetto chiave che sta alla base si questo meccanismo ovvero il Cluster.
In informatica un computer cluster, o più semplicemente un cluster (dall'inglese grappolo), è un insieme di computer connessi tra loro tramite una rete telematica. Lo scopo di un cluster è quello di distribuire un'elaborazione molto complessa, tra i vari computer che ne fanno parte. In sostanza, un problema che richiede molte elaborazioni per essere risolto viene scomposto in sottoproblemi, i quali vengono risolti in parallelo tra le varie macchine componenti il cluster in questione, aumentando così la potenza di calcolo del sistema. Partendo da questo concetto possiamo dire che il load balancing, in italiano bilanciamento del carico, è una tecnica informatica che consiste nel distribuire il carico di uno specifico servizio, ad esempio la fornitura di un sito web, tra più server. Attraverso tale tecnica, in sostanza, diverse macchine facenti parte di un unico cluster, collaborano in modo ottimale nel far fronte alle richieste di elaborazione, distribuendo il lavoro tra i diversi sistemi collegati in Rete. In questo modo, si aumentano la scalabilità e l'affidabilità dell'architettura nel suo complesso. La scalabilità deriva dal fatto che, nel caso sia necessario, si possono aggiungere nuovi server al cluster in pochissimo tempo, mentre la maggiore affidabilità è dovuta al fatto che la rottura di una delle macchine non comprometterebbe la fornitura del servizio, resa possibile grazie alla possibilità di rimpiazzare quest'ultima in brevissimo tempo; non a caso i sistemi di load balancing in genere integrano dei sistemi di monitoraggio, come visto precedentemente, che escludono automaticamente dal cluster i server non raggiungibili e/o guasti, evitando così il fallimento e il conseguen-

te rifiuta di una porzione delle richieste in input. Grazie a questa breve definizione siamo riusciti a focalizzare l'attenzione sull'importanza che questa tecnica possiede e del ruolo critico che svolge nel sistema[11], poiché è responsabile dello smistamento delle richieste dell'utente tra i vari server affinchè queste vengano elaborate nel minor tempo possibile.

1.2.2 AMAZON EC2

Amazon EC2[12] è il cuore della nuvola di Amazon e fa parte del pacchetto AWS(Amazon Web Services). Esso fornisce le API necessarie per la gestione, il rilascio e la creazione di server virtuali, in pratica tramite EC2, ovunque siamo, è sufficiente una connessione ad Internet per creare un server virtuale sulla nuvola di Amazon. Questo significa che quando si ha la necessità di creare un server virtuale nell'ambiente Amazon, basta generare un nuovo nodo basato su un immagine (AMI, Amazon Machine Imagine), comprendende una serie di software e un SI, e parametrizzarla a proprio piacimento. Una volta capito questo breve procedimento è sufficiente ripetere questa operazione più volte per creare la nostra rete virtuale di server o Virtual Cluster.
Le caratteristiche principali di EC2 sono:

- EBS (Amazon Elastic Block Store): offre uno storage per le istanze indipendentemente dalla durata della loro vita.
- Multi Locazione: per garantire continuità da fallimenti del Data Center e ottenere una miglior latenza è possibile eseguire istanze in più data center posti fisicamente in posti strategici molto lontani tra loro.
- Indirizzo IP Elastico: l'indirizzo IP viene associato all'account e non alle singole istanze, così facendo il possessore lo gestisce fino alla richiesta esplicita di rilascio. In questo modo in caso di fallimento dell'istanza il problema è circonciso, garantendo cosi fault tolerance sulle istanze.
- VPC (Amazon Virtual Private Cloud): permette, come visto, di creare una rete virtuale e configurarne le politiche di sicurezza.

- Amazon Cloud Watch: strumento utile alla misurazione delle prestazioni del sistema. Permette la visualizzazione delle performance, l'utilizzo delle risorse e del flusso delle richieste, il tutto tramite grafici e statistiche.
- Auto Scaling: attraverso politiche basate sullo scelte dell'utente l'auto scaling riduce i costi (scale-down) e migliora le prestazioni (scale-up).
- HPC (High performance Computing) cluster: clienti con carichi di lavoro molto pesanti possono godere di un infrastruttura a loro dedicata e personalizzata in base alle esigenze. Questa soluzione è creata ad Hoc per la ricerca scientifica data la disponibilità di elevate prestazioni.

L'ultima caratteristica è l' Elastic Load Balancing a cui dedichiamo una parentesi a parte. L'Elastic Load Balancing è una tecnica, compresa nel pacchetto EC2, che permette il bilanciamento delle richieste di applicazioni in entrata su più istanze di Amazon EC2. Questo permette di raggiungere ottime prestazioni e una buona faultolerance all'interno della nostra rete virtuale, grazie alla capacità di riconoscere le istanze "difettose" così da eliminarle dalle possibili destinazioni delle richieste in ingresso. Possiamo decidere di applicare il bilanciamento sia in una singola Availability Zone o ad un gruppo di esse per un mantenimento del livello prestazionale ancora più elevato. Il tutto è configurabile dall'utente che può decidere, a seconda delle sue esigenz, la configurazione migliore della politica di suddivisione del carico così da ottenere le migliori prestazioni possibili.

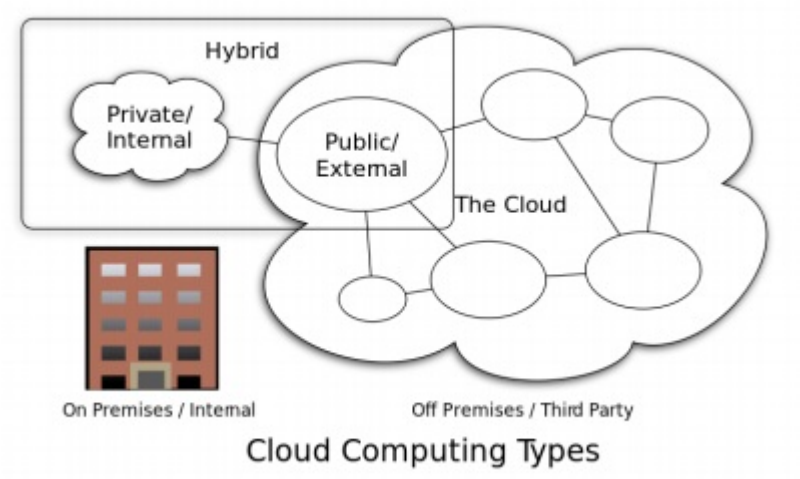

Figura 3: Modelli di Deployment

1.3 PROBLEMATICHE LEGATE AL CLOUD

In questo capitolo ci occuperemo di tutte le problematiche etiche, sociali, economiche e informatiche che accompagnano l'utilizzo di un servizio di Cloud Computing. A tal proposito, è utile, in primo luogo, partire con una descrizione delle possibili modalità di deployment del sistema di Cloud, infatti a seconda della scelta non variano solamente parametri, caratteristiche, pregi e difetti ma anche i problemi che ne derivano.

1.3.1 MODELLI DI DEPLOYMENT

Quando si offre una soluzione di cloud computing, è indispensabile decidere quale modello vogliamo implementare a seconda delle nostre necessità. Ci sono quattro tipologie tra cui scegliere: public cloud, private cloud, community cloud e hybrid cloud (Figure 3). Questa suddivisione, adottata anche in "Cloud Computing. Distributed Internet Computing for IT and Scientific Research" (2009)[13] si compone delle seguenti categorie:

1. Public Cloud
   Il public cloud permette agli utenti di accedere al cloud tramite web browser. Gli utenti pagano solo per il tempo di utilizzo

del servizio, ma hanno garantito un livello di sicurezza minore rispetto agli altri modelli, poichè il mezzo di comunicazione utilizzato (internet) è intrinsecamente insicuro. La maggior parte dei problemi di sicurezza, in questa tipologia di deployment cloud, nasce dal fatto che il cliente si rivolge a terzi per la gestione dei dati, perdendone così il controllo fisico. Al momento della stipula del contratto saranno necessari vari accordi sugli standard di sicurezza e una suddivisione precisa e chiara de ruoli e delle responsabilità tra cliente e provider al fine di preservare i dati.

2. Private Cloud
   Una soluzione di tipo private cloud prevede il deploy di un cloud all'interno dei confini di una azienda. Il vantaggio principale è la facilita di gestione della sicurezza, della manutenzione e degli aggiornamenti. Rispetto ai cloud pubblici, dove tutte le risorse e le applicazioni sono gestite dal Cloud Provider e condivise da tutti gli utenti, nella tipologia privata infatti, i servizi sono messi a disposizione solamente agli user aziendali e le risorse sono gestite dall'organizzazione stessa. La sicurezza risulta così piu elevata e i problemi da essa derivanti molto minori. A fronte di questi vantaggi bisogna annoverare però che, adottando questa tipologia di soluzione, viene a mancare uno dei vantaggi principali del cloud, l'essere indipendenti dalla costruzione e dalla manutenzione di un'infrastruttura di computing. L'azienda che decide di adottare un modello di deployment di tipo privato dovrà infatti spendere considerevoli risorse per implementarlo.

3. Community Cloud
   Nel community cloud le infrastrutture sono condivise tra le diverse organizzazioni di un gruppo specifico aventi interessi "informatici" comuni. Tali interessi potrebbero essere legati alla conformità normativa, ai requisiti di rendimento e così via. Le infrastrutture possono essere on-premises (ovvero poste all'interno della sede) o off-premises. I costi sono divisi tra meno utenti rispetto ad un cloud pubblico, però la sicurezza è molto maggiore, in quanto gli accessi sono limitati agli utenti della Community.

4. Hybrid Cloud
   L'ultimo modalità di deployment che vediamo è quello che viene considerato un modello ibrido in quanto composto da due o più cloud facenti parte delle precedenti categorie, proprio come in Figure 3. Così facendo l'utente finale potrà godere dei pregi di tutte le modalità implementate nel sistema hybrid infatti, pur essendo considerate come entità singole, quello che si viene a creare è un modello di deployment multiplo. Di contro le architetture di questo tipo necessitano sia di risorse ospitate in locale che di una infrastruttura cloud off-site. Quello che succede, in genere, è che l'infrastruttura privata viene utilizzata per smaltire il fabbisogno standard dell'azienda ed in caso di picchi di necessità si va ad attivare il Public Cloud ottenendo così risorse computazionali extra.

Da questa suddivisione in tipologie di deployment abbiamo capito che la conseguente scelta è un aspetto molto importante quando si decide di usufruire di un servizio cloud. A seconda del modello scelto cambia infatti il grado di sicurezza dei nostri dati. Proprio questo aspetto è uno dei problemi principali che i critici usano per declassare la scelta di un'architettura Cloud rispetto ad altre strategie.
Utilizzare un servizio di Cloud Computing per memorizzare dati personali o sensibili, espone l'utente a potenziali problemi di violazione della privacy. I dati personali vengono memorizzati nelle Server Farms di aziende che spesso risiedono in uno stato diverso da quello dell'utente. Il cloud provider, in caso di comportamento scorretto o malevolo, potrebbe accedere ai dati personali per eseguire ricerche di mercato e profilazione degli utenti. Con i collegamenti wireless poi, il rischio aumenta essendo maggiormente esposti a problematiche riguardanti la pirateria informatica, a causa della minore sicurezza offerta dalle reti senza fili. In presenza di atti illegali, come appropriazione indebita o illegale di dati personali, il danno potrebbe essere molto grave per l'utente, con difficoltà di raggiungere soluzioni giuridiche e/o rimborsi se il fornitore risiede in uno stato diverso da paese dell'utente. Pensando poi alle industrie, tutti i dati memorizzati nelle memorie esterne sono seriamente esposti a eventuali casi di spionaggio industriale con possibile conseguente perdita di ingenti somme di denaro. Garantire la sicurezza dei dati per i Cloud Provider oggi, è tanto difficile quanto importante e scoglio necessario da superare al fine di ottenere la fiducia del cliente

finale. Strettamente collegato alla sicurezza è il problema della proprietà dei dati: se l'azienda che offre i servizi Cloud è considerata proprietaria dei dati posti all'interno della nuvola, allora essa è soggetta a precisi diritti e doveri, tra cui il controllo dei contenuti. Se l'azienda invece è considerata solamente "custode" delle informazioni allora i diritti e i doveri cambiano in maniera radicale.
Sempre legati alla sicurezza sono i problemi internazionali di tipo economico e politico derivati dal Cloud. Questi possono verificarsi quando dati pubblici sono raccolti e conservati in archivi privati, situati in un paese diverso da quelli degli utenti della "nuvola". Produzioni cruciali e di carattere intellettuale, insieme a una grande quantità di informazioni personali sono memorizzate crescentemente in forma di dati digitali in archivi privati centralizzati e parzialmente accessibili. Nessuna garanzia viene data agli utenti per un libero accesso futuro.
A carattere economico invece, sono legate problematiche dovute alla localizzazione degli archivi della "nuvola" in alcuni paesi ricchi. Se non regolato da specifiche norme internazionali ciò potrebbe:

- Aumentare il "digital divide" tra paesi ricchi e poveri (se l'accesso alle conoscenze memorizzate non sarà liberamente garantita a tutti)
- Favorire principalmente grandi corporation con "organismi policentrici" e "menti monocentriche" dislocate principalmente nei Paesi della "nuvola", essendo la proprietà immateriale considerata come un fattore strategico per le moderne economie "knowledge-based". Maggiori sicurezze e garanzie vi sono nel caso in cui il fornitore del servizio appartenga alla stessa nazione/area applicando le medesime leggi/normative sulla privacy e sicurezza del cliente (la legislazione USA o di altre nazioni è molto diversa dall'italiana e diventa impossibile pensare di soddisfare normative nazionali con servizi in cloud di altre nazioni).

A livello informatico inoltre, è cruciale per lo user avere la sicurezza della continuità del servizio "acquistato" infatti, delegando un'azienda esterna per la gestione dei dati e la loro elaborazione, l'utente si troverà fortemente limitato nel caso in cui i suddetti servizi non risulteranno operativi (out of service). Un eventuale malfunzionamento

inoltre colpirebbe un numero molto elevato di persone contemporaneamente, essendo questi servizi condivisi. Anche se i migliori Cloud Provider utilizzano architetture ridondate e personale qualificato al fine di evitare malfunzionamenti dei sistema e ridurre la probabilità di guasti visibili dall'utente finale, non eliminano del tutto il problema.
Non bastassero tutta questa serie di problematiche, infine, esistono problematiche legate alla difficoltà di migrazione dei dati nel caso di un eventuale cambio del gestore dei servizi cloud. Ad oggi non esistono standard definiti tra i gestori dei servizi che regolamentano un eventuale switching di operatore che risulterebbe estremamente complesso. Questo è una problematica tutt'altro che remota nel verificarsi, basti pensare al caso in cui il gestore di servizi cui ci siamo affidati cada in fallimento.
A fronte di tutte queste problematiche l' elemento essenziale però, è indubbiamente la fiducia. L'utente, più o meno tecnologico, deve fidarsi in ogni caso dell'hardware, del sistema operativo, del software, del Cloud provider e così via, se si vuole garantire un progresso tecnologico legato alla società. Un eventuale crash del sistema, un errore nel software, sono elementi che possono minare la sicurezza dei dati, come visto e con la scelta di una strategia basata su cloud computing si aggiunge un ulteriore elemento di preoccupazione. Fino al momento in cui il computer è all'interno della propria rete, si hanno tutti i mezzi per proteggerlo mentre se decidiamo di affidarci a servizi di cloud computing, non vi è altra possibilità che fidarsi di chi offre il servizio.

2 SIMULAZIONE AD EVENTI DISCRETI

In questo capitolo affronteremo il mondo della simulazione, in particolare quella ad eventi discreti. Questo sarà possibile grazie ad una serie di definizioni che daranno un'infarinatura generale sul tema, per poi addentrarci nell'ala statistica della simulazione necessaria ad avvalorare i risultati che otterremo in fase di progettazione vera e propria.

2.1 SISTEMA AD EVENTI DISCRETI

Un sistema è una collezione di componenti dipendenti, le cui azioni sulle altre formano un processo dinamico. Per catturarne il comportamento è necessario costruire un modello appropriato delle sue rappresentazioni interne e un insieme di regole di trasformazione. Le rappresentazioni interne individuano delle variabili che corrispondono ad attributi del sistema, a sue caratteristiche o a sue proprietà. Queste variabili si chiamano *variabili di stato* e definiscono lo stato del sistema. A loro volta le variabili di stato possono cambiare in maniera continua o discreta e quando ciè avviene si dice che si è verificato un *evento.* Esso può essere *endogeno* se si verifica all'interno del sistema o *esogeno* se lo influenza. Secondo un altra possibile differenziazione il sistema può essere:

- Time Event: se gli eventi sono individuati da un istante temporale nel calendario degli eventi
- State Event: se gli eventi sono individuati da una condizione che specifica delle circostanze (diverse dal tempo) sotto cui il sistema cambia stato

I sistemi ad eventi discreti, anche detti DES (Figure 4), differiscono dai sistemi dinamici classici, in quanto l'evoluzione del loro stato, non dipende dal tempo (*time driven*), bensì dal susseguirsi di un insieme finito di eventi (*event driven*). Essi sono caratterizzati da una serie di ingressi e da una serie di uscite. Nel mezzo avviene la trasformazione secondo le regole dettate da chi ha progettato il sistema. Da qui deriva un concetto chiave della simulazione ovvero lo stato.
Si definisce stato di un sistema dinamico $x(t_0)$ all'istante t_0, l'insieme delle informazioni necessarie affinchè l'uscita $y(t)$, per $t \geq t_0$, sia univocamente determinata dall'ingresso $u(t)$ con $t \geq t_0$.

Per avere una visione più completa delle caratteristiche dei DES però, dobbiamo introdurre altre due calssificazioni fondamentali in ambito di simulazione. La prima afferma che se l'uscita rimane invariata a parità di condizioni iniziali e ingresso, allora il sistema si dice *tempo invariante*, altrimenti *tempo variante*.
La seconda diversivicazione si basa sul concetto di sovrapposizione degli effetti. Si dice che un sistema dinamico è *lineare* se:

- gli insiemi degli ingressi U, delle uscite Y e degli stati X, sono spazi lineari (o vettoriali)
- esiste una rappresentazione esplicita

$$x(t) = \phi(t, t_0, x_0, u)$$

$$y(t) = \eta(t, x(t), u(t))$$

tale che:
$\phi(t, t_o, k_1 x_{01} + k_2 x_{02}, k_1 u_1 + k_2 u_2) = k_1 \phi(t, t_0, x_{01}, u_1) + k_2 \phi(t, t_0, x_{02}, u_2)$
$\eta(t, k_1 x_1(t) + k_2 x_2(t), k_1 u_1(t) + k_2 u_2(t)) = k_1 \eta(t, x_1(t), u_1(t)) + k_2 \eta(t, x_2(t), u_2(t))$
Dopo aver dato questa serie di classificazioni possiamo dire che un

DES è un sistema dinamico, tempo invariante, non lineare, event driven a stati discreti (Figure 4)

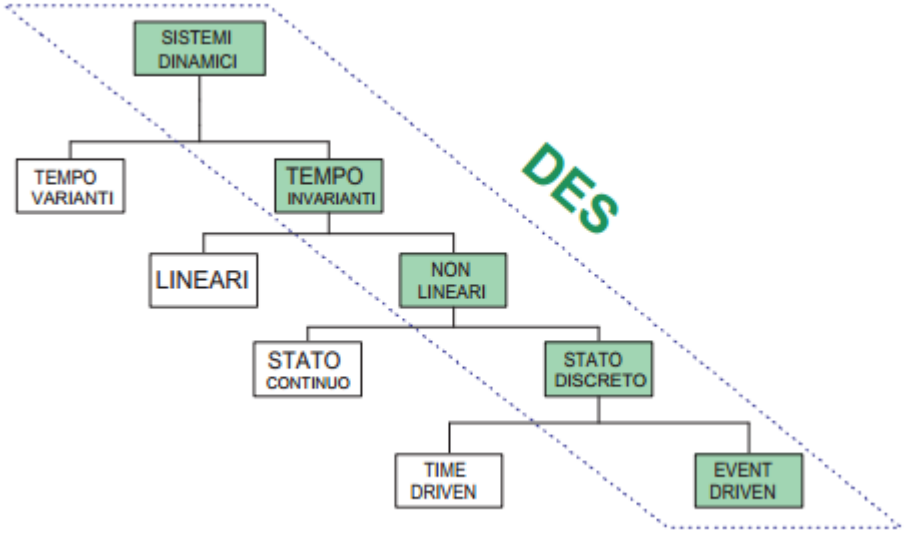

Figura 4: DES

Prima di passare alla sezione successiva andiamo a definire un oggetto che rappresenta il più semplice sistema ad eventi discreti, ovvero la coda singola (Figure 5). Tale componente è la base per la costruzione di sistemi più complessi, tra cui il modello di simulazione che verrà utilizzato nel progetto di tesi.

Figura 5: Modello a Single Queue

Questa tipologia di componente prevede una serie di arrvi e partenze, il tutto regolato da una politica, in questo progetto di tesi FIFO (First In First Out), che gestisce il passaggio da Input a Output. Le trasformazioni che avvengono al suo interno sono gestite e programmate da chi sviluppa la coda ed è importante ricordare che la

coda deve essere sempre attiva quando vi è un cliente in attesa del servizio. Un parametro fondamentale in questo ambito è il numero totale di clienti nel sistema coda che identifichiamo con $n'(t)$dove

$$n'(t) = n(t) + \delta(t)$$

è dato dalla somma dei clienti nella fila di attesa e del cliente in servizio, se presente.
Le azioni che vengono effettuate all'arrivo e alla partenza di un nuovo cliente sono elencate di seguito e schematizzate nei diagrammi di flusso corrispondenti (Figure6, Figure7):

Figura 6: Arrivo Cliente

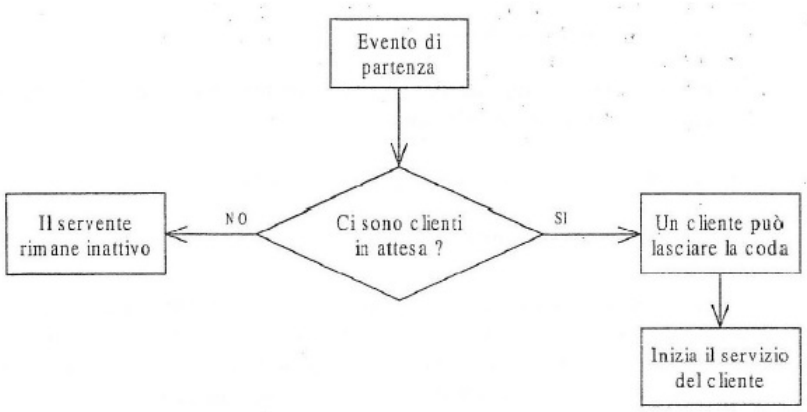

Figura 7: Partenza Cliente

2.2 SIMULAZIONE

In questo capitolo descriveremo la simulazione, definendo tutti i suoi componenti e le caratteristiche che la contraddistinguono. Ci avvicineremo così al progetto vero e proprio di questa tesi, che verrà poi descritto nel capitolo successivo, introducendo alcune definizioni ed alcuni elementi che vedremo essere chiave nel proseguo.
Dopo questa premessa possiamo dare la definizione di simulazione introdotta da Stewart Robinson [14] dicendo che la simulazione ovvero è la rappresentazione della realtà che permettere di valutare e prevedere lo svolgersi dinamico di una serie di eventi, susseguenti all'imposizioni di determinate condizioni controllate dall'analista o dall'utente. In altre parole:
"Con un programma di simulazione si riproduce, in un ambiente controllato, il funzionamento di un sistema reale al fine di analizzare il suo comportamento nelle diverse condizioni possibili. Per lo studio dell'evoluzione dinamica del sistema va comunque sviluppato un modello, che in questo ambito è ovviamente un modello simulativo."
(Di Febbraro A., Giua A., 2002) [15]
I dati in output di una simulazione vengono spesso rielaborati, tramite calcoli statistici e trasformati in informazioni utili al progettista e/o analista. Così facendo è possibile ottenere previsioni comportamentali, stime numeriche delle grandezze fisiche in gioco e così via. Ovviamente tanto più il modello di realta da simulare sarà complesso, tanto maggiore sarà il numero di informazioni ottenute. Il prezzo da pagare per tale completezza e varietà è ovviamente il tempo; le operazioni di simulazione sono infatti assai lunghe, affinchè si possano ottenere dei dati sufficientemente sensati e tali da dare la possibilità di ottenere un modello della realtà valido e rappresentativo. Inoltre il raggiungimento di dati statisticamente significativi spesso richiede l'esecuzione di varie ripetizioni (run) di tali simulazioni e il contributo di personale esperto.
A loro volta però, i vantaggi della simulazione sono diversi:

- lo stesso modello può essere utlizzato più volte, anche per realtà differenti
- lo studio che precede la simulazione permette di acquistare conoscenza sul sistema

- i modelli simulativi spesso sono gli unici applicabili, in quanto una realizzazione reale di tali modelli potrebbe richiedere molto tempo e denaro
- spesso tali modelli non richiedono eccessive assunzioni semplificative

2.2.1 PROCESSO DI SIMULAZIONE

Al fine di ottenere un modello di simulazione funzionante e statisticamente correto è opportuno procedere con la seguente serie di passi:

1. Definizione degli obiettivi e delle problematiche da esaminare: un' attenta analisi del problema consente di circoscriverne l'esame riducendo il successivo tempo di analisi
2. Stesura di un modello concettuale: consiste nella comprensione e modellazione del sistema produttivo che si intende simulare; questa fase è particolarmente importante in quanto definirà il comportamento dei diversi flussi di materiale e di informazioni che attraverseranno il modello
3. Validazione del modello concettuale: si tratta di un confronto con la direzione dell'impresa e con gli operatori per assicurarsi della capacità del modello di offrire un immagine consistente della realtà
4. Analisi dei dati in ingresso: la raccolta e l'analisi dei dati che diverranno la base per la definizione dei parametri di funzionamento del sistema. Attraverso le tecniche del calcolo delle probabilità diviene possibile definire una distribuzione di probabilità per ogni parametro, da inserire all'interno del modello. Questa è una fase cruciale in quanto, tanto più i dati in input saranno mirati e sensati, tanto più otterremo un output soddisfacente in termine di precisione e di quantitativo di informazioni
5. Scrittura del modello in termini matematici
6. Calibrazione e valutazione
7. Definizione di un piano degli esperimenti: una singola iterazione ("run") di simulazione non ha alcun significato; rappresenta

solo una delle possibili evoluzioni del sistema. È quindi opportuno effetture diversi "run" per poi analizzare i parametri in uscita. La lunghezza della singola iterazione e il numero delle iterazioni vengono determinate in questa fase

8. Analisi dei dati in uscita: dopo aver raccolto i dati relativi ai parametri, depurati da eventuali transitori è possibile creare degli intervalli di confidenza ovvero stimare il "range" di valori in cui i parametri che analizzano il problema proposto al primo passaggio possono oscillare

Tutti questi passi sono però scanditi nel tempo, sebbene il modello sia event-based, e spesso non è facile districarsi nella gestione, in fase di implementazione, della linea temporale degli eventi. Per questo motivo è utile definire le tre diverse connotazioni di avanzamento del tempo nella simulazione, ovvero il tempo reale, il tempo simulato e il tempo di esecuzione. Nel primo caso la visione del tempo è continua e il focus è sul sistema reale mentre nel secondo l'attenzione è posta sul modello e l'avanzamento è per eventi. Nel terzo e ultimo caso il punto di vista è del processore e viene identificato il tempo di esecuzione del programma "simulatore". Lo schema simulativo "event-oriented" infatti, procede producendo una serie di "fotografie" del sistema in istanti di tempo discreti: tali istanti di tempo sono le occorrenze degli eventi e ogni fotografia contiene al suo interno lo stato del sistema, una lista degli eventi già schedulati e i contatori progressivi per il calcolo delle statistiche.
Il meccanismo che permette di far avanzare correttamente la simulazione e garantire che tutti gli eventi avanzino nell'ordine cronologico corretto si basa sulla costruzione e sulla scansione di una lista di eventi attivi chiamata LEA (Lista Evemti Attivi). Questa lista contiene tutti gli eventi già schedulati per accadere nel futuro con associati i corrispondenti tempi di occorrenza (Figure 8), ed è ordinata in base ad essi.

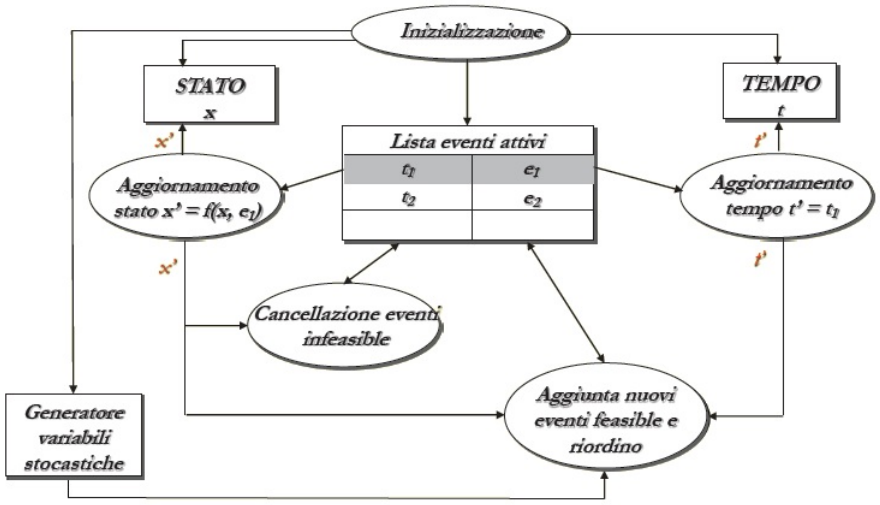

Figura 8: Descrizione LEA Process

Grazie ad una serie di passi, ripetuti iterativamente, è possibile gestire gli eventi e simularli secondo una linea temporale in modo corretto. Questa viene chiamata procedura di simulazione ed è composta dai seguenti passi:

1. Rimozione del primo elemento di LEA (t_1, e_1)
2. Aggiornamento del tempo di simulazione a t_1
3. Aggiornamento dello stato del sistema a $x' = f(x, e_1)$ dove f rappresenta la transizione di stato relativa all'evento e_1
4. Cancellazione da LEA degli eventi resi non feasible dall'evento e_1
5. Aggiunta in LEA degli eventi resi feasible dall'evento e_1
6. Ordinamento di LEA in ordine cronologico crescente

La simulazione inizia con la fase di inizializzazione di LEA dove vengono inseriti gli eventi possibili all'istante 0 e termina quando in cima a LEA è presente l'evento *fine simulazione*, inserito in coda o in fase di inizializzazione o al raggiungimento di particolari condizioni. Per fare in modo che tutto questo funzioni in maniera ottimale, l'implementazione di LEA deve essere eseguita garantendo la massima

efficienza sia delle operazioni di inserimento, cancellazione, modifica e ricerca di un elemento, sia del riordino della lista.

2.2.2 STIMA STATISTICA DEI DATI IN OUTPUT

Per garantire dati statisticamente significativi, come detto, occorre ripetere la simulazione più volte, al fine di verificare quante più possibili combinazioni di eventi. Ogni singola iterazione, all'interno della simulazione, è detta run e genera dei dati in output, i quali devono essere sottoposti ad un'attenta analisi che consiste nella determinazione (stima) delle grandezze obiettivo. La stima di tali grandezze affonda nella statistica e si basa su tecniche di stima di parametri. Queste grandezze non sono altro che parametri statistici di processi stocastici che dipendono dalle condizioni iniziali della simulazione, dalla sua durata e dalla sequenza di numeri random utilizzati.
Sia θ una delle grandezze (parametri) da determinare allora, date $X_1, X_2...X_n$ osservazioni di tale grandezza, si vuole determinare una stima di θ. Tale stima può essere, puntuale se si desidera un valore plausibile di teta, oppure intervallare se si desidera una stima dell'intervallo di valori in cui θ cade con una probabilità p definita a priori.

Stima per punti o Puntuale Sia $X_1, X_2...X_n$una sequenza di campioni indipendenti ed identicamente distribuiti, ossia la sequenza è costituita da campioni di un unico processo stocastico e il parametro da stimare θ è il valor medio di tale processo, lo stimatore per punti di θ è:
$\hat{\theta}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$
Di tale parametro è possibile poi calcolare anche la stima della varianza (e di conseguenza la deviazione standard) attraverso la varianza campionaria nel seguente modo:
$\hat{\sigma}^2(\hat{\theta}_n) = \frac{S_n^2}{n} = \frac{1}{n}\sum_{i=1}^{n} \frac{(X_i - \hat{\theta}_n)^2}{n-1}$
Una volta calcolati questi 2 parametri è quindi possibile avere una stima del paramentro obiettivo teta.

Stima Intervallare con Varianza Nota Supponendo che i campioni estratti provengano da una popolazione distribuita secondo

una normale di parametri $N(\theta, \sigma^2)$, con σ noto, allora l'intervallo di confidenza per θ di livello α è dato da:
La costruzione di un intervallo di confidenza per θ sotto l'assunzione di varianza nota, si basa sul seguente risultato:
la media campionaria

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

è una variabile aleatoria che si distribuisce approssimativamente come una normale $N(\theta, \frac{\sigma^2}{n})$ e tale approssimazione migliora all'aumentare della dimensione campionaria n. Se dunque usiamo la media campionaria come stimatore della media della popolazione, il fatto che la sua distribuzione sia centrata sul valore vero del parametro θ indica che $\overline{x}$ uno stimatore non distorto. Inoltre, il rapporto $\frac{\sigma^2}{n}$ misura la precisione dello stimatore: come ci si potrebbe aspettare, tale precisione è tanto minore quanto più elevata è la varianza σ^2 e tanto maggiore quanto più elevata è la dimensione campionaria n. Dal fatto che $\overline{x} \sim N(\theta, \frac{\sigma^2}{n})$, si deduce che

$$\frac{\overline{x} - \theta}{\sqrt{\frac{\sigma^2}{n}}} \sim N(0, 1)$$

Per ogni valore di probabilità $1 - \alpha$, possiamo quindi scrivere che

$$P(-z_{\frac{\alpha}{2}} \leq \frac{\overline{x} - \theta}{\sqrt{\frac{\sigma^2}{n}}} \leq z_{\frac{\alpha}{2}}) = 1 - \alpha$$

dove $z_{\frac{\alpha}{2}}$ è il quantile della normale di ordine $1 - \frac{\alpha}{2}$, ovvero il punto che si lascia a sinistra un'area sotto la normale pari a $1 - \frac{\alpha}{2}$. Ad esempio se $1 - \alpha = 0.95$, allora $z_{\frac{\alpha}{2}} = 1.96$.
Un intervallo di confidenza può essere quindi costruito, dopo una serie di passaggi, ottenendo così il seguente risultato:

$$\overline{x} - z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}} \leq \theta \leq \overline{x} + z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}$$

In altre parole, è approssimativamente uguale a $1 - \alpha$ la probabilità che i due estremi dell'intervallo

$$(\overline{x} - z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}, \overline{x} + z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}})$$

contengano il valore "vero" della media θ della popolazione.

Stima Intervallare con Varianza incognita Nella maggior parte delle applicazioni, è difficile avere una stima attendibile della varianza σ^2 della popolazione e si preferisce, in genere, stimarla sulla base del campione estratto. Una stima non distorta della varianza della popolazione è data da
$\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{n}{n-1}(\frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2)$
che non è altro che la varianza campionaria corretta dal fattore $\frac{n}{n-1}$. Tale correzione dipende dal fatto che, per piccoli campioni, la varianza campionaria è uno stimatore distorto della varianza della popolazione, cioè la sua distribuzione campionaria non ha come valore atteso il valore vero del parametro σ^2 . Per grandi campioni, il fattore di correzione $\frac{n}{n-1} \approx 1$ e dunque l'uso della varianza campionaria fornisce stime attendibili della varianza della popolazione.
In questo caso, per costruire un intervallo di confidenza della media θ della popolazione, occorre utilizzare il fatto che la distribuzione della variabile aleatoria

$$\frac{\overline{x} - \theta}{\sqrt{\frac{\hat{\sigma}^2}{n}}}$$

segue approssimativamente quella di una t di Student con $n-1$ gradi di liberta, dove n è la dimensione del campione estratto e che tale approssimazione migliora all'aumentare di n. La distribuzione t di Student è molto simile a quella di una normale standardizzata. Essa è infatti centrata sullo 0 e simmetrica rispetto ad esso. Si differenzia dalla distribuzione normale in quanto ha delle code "più" pesanti, ovvero valori lontani dallo 0 hanno una probabilità di essere estratti più elevata di quella che avrebbero avuto se fossero stati estratti da una normale standardizzata. Tali differenze si attenuano sempre più all'aumentare della numerosità campionaria, per cui quando n è molto elevato, si può utilizzare la distribuzione normale standardizzata in luogo della t.
La costruzione dell'intervallo di confidenza segue linee analoghe a quelle mostrate nella sezione precedente. Si indichi pertanto con $t_{n-1,\frac{\alpha}{2}}$ il quantile di ordine $1 - \frac{\alpha}{2}$ di una t di Student di $n-1$ gradi di libertà, ovvero il punto che si lascia a sinistra un'area sotto la t

pari a $1 - \frac{\alpha}{2}$. Un intervallo di confidenza può allora essere costruito sulla base della seguente equazione:
$1 - \alpha = P(\bar{x} - t_{n-1,\frac{\alpha}{2}} \sqrt{\frac{\hat{\sigma}^2}{n}} \leq \theta \leq \bar{x} + t_{n-1,\frac{\alpha}{2}} \sqrt{\frac{\hat{\sigma}^2}{n}})$
In altre parole, è approssimativamente uguale a $1 - \alpha$ la probabilità che i due estremi dell'intervallo
$(\bar{x} - t_{n-1,\frac{\alpha}{2}} \sqrt{\frac{\hat{\sigma}^2}{n}},\ \bar{x} + t_{n-1,\frac{\alpha}{2}} \sqrt{\frac{\hat{\sigma}^2}{n}})$
contengano il valore "vero" della media θ della popolazione.
All'interno del nostro progetto, queste nozioni statistiche serviranno a determinare se l'esperimento che stiamo conducendo ha un numero n di campioni sufficientemente grande da garantire significatività statistica.

3 OMNET++

In questo capitolo, dopo una breve introduzione generale riguardante i linguaggi di programmazione per la simulazione, analizzeremo il software utilizzato in questo progetto ovvero OMNET++. Per prima cosa proporremo una carrellata generale sulle caratteristiche di tale prodotto, analizzandone i pregi e i difetti che lo rendono il SW per la simulazione più utilizzato in ambito accademico. Secondariamente poi, ne descriveremo i componenti principali, le modalità di programmazione, l'architettura ed infine, i passi necessari ad una corretta installazione e configurazione attraverso l'IDE Eclipse.

3.1 LINGUAGGI DI PROGRAMMAZIONE PER LA SIMULAZIONE

In questa prima parte analizzeremo gli SPL, ovvero i linguaggi di programmazione per la simulazione. Questi software sono nati allo scopo di velocizzare lo sviluppo del modello, la rappresentazione dei dati di ingresso e l'analisi dei risultati e grazie a queste caratteristiche sono riusciti ad incrementare l'utilizzo della simulazione come strumento di analisi, favorito anche dalla diminuzione progressiva dei costi di sviluppo di un loro modello. Per operare la scelta del simulatore, è necessario considerare un insieme di caratteristiche generali:

- Tipo di licenza: indica se il simulatore è free software o è disponibile solo in versione commerciale
- Linguaggio dei componenti dei nodi: indica il linguaggio di programmazione (general purpose) usato per implementare i componenti dei nodi
- Tipo di simulazione: indica se la simulazione è ad eventi discreti o tempo continuo
- Emulazione: indica se è supportata l'emulazione
- Simulazione parallela: indica se è presente un supporto all'esecuzione parallela in multithreading e se è consentito all'utente, configurare tale funzionalità mediante la gestione dei thread.

Una classificazione di maggior dettaglio poi, riguarda le caratteristiche qualitative, che illustrano in maniera precisa il modo in cui

è possibile condurre la simulazione. Le caratteristiche qualitative sono ad esempio:

1. Interfaccia grafica: rappresenta la possibilità di avere e sfruttare un'adeguata GUI per l'avvio, l'esecuzione e la terminazione della simulazione
2. Estensibilità e la modificabilità del prodotto: questi parametri specificano la possibilità di estendere e modificare i moduli del simulatore specializzando le funzionalità in base alle proprie esigenze di modellazione
3. Modularità: indica la possibilità di comporre tra loro le entità della simulazione, per formare entità più generali
4. Tipologia di guasti: indica il tipo di guasto che l'utente può simulare durante la simulazione. L'attenzione è stata rivolta alla perdita dei pacchetti ed alla corruzione dei bit dei pacchetti
5. Statistiche finali: garantisce la possibilità di produrre agevolmente statistiche finali, creare grafici, esportare dati su fogli di lavoro e eseguire report standard e/o personalizzati

Oltre a questi parametri, durante la scelta del software di simulazione più adatto alle nostre esigenze, bisogna anche considerare le caratteristiche relative alla capacità di elaborazione come la velocità, la capacità di generare numeri casuali, la gestione dei dati, le librerie disponibili, la gestione di simulazioni non terminanti e di replicazioni indipendenti e cosi via.
Un'analisi accurata di tutti i parametri presenti in questa introduzione porta alla scelta dell'SPL che più si avvicina alle nostre esigenze.
Negli ultimi 20 anni sono stati sviluppati numerosi prodotti tra cui scegliere, sia per scopi generali, sia per applicazioni specifiche. Attualmente tali prodotti sono molto diffusi e disponibili solitamente anche in versioni per PC comuni, senza particolari requisiti di sistema. In questo progetto di tesi abbiamo deciso di utilizzare un software free, molto utilizzato in ambito accademico, ovvero OMNET++ [16] per una serie di caratteristiche che ora vediamo nel dettaglio.

3.2 CARATTERISTICHE DI OMNET++

OMNeT++ (Objective Modular Network Testbed in C++) è un simulatore basato sul linguaggio C++, ad eventi discreti, per modelli di reti di comunicazione[17]. È un prodotto open-source nato nel 2003, che può essere usato sotto licenza GNU, ovvero senza scopi di lucro, e che viene principalmente utilizzato in ambito di ricerca accademica. Una delle sue principali caratteristiche è la modularità dei suoi componenti, ovvero ogni entità presente è rappresentata mediante un modulo, e tutti i moduli sono organizzati gerarchicamente e comunicano tra loro tramite scambio di messaggi che viaggiano attraverso porte e connessioni. Esso è classificato fra i simulatori comportamentali, ovvero che consentono la simulazione su ogni nodo della rete di tutti gli strati dello stack protocollare, al fine di studiare il traffico dei messaggi sulla rete ed il comportamento complessivo del modello. Il simulatore OMNeT++ è anche Component-Oriented, la sua architettura infatti, è costituita da componenti che, interagendo tra loro, consentono l'esecuzione del programma di simulazione con la finalità di supportare la simulazione di reti anche di grandi dimensioni. I suoi moduli ottimamente realizzati e totalmente riutilizzabili, sono combinabili in vari modi al fine di ottenere la configurazione desiderata. Fra le varie caratteristiche che lo rendono un ottimo simulatore vi è la facilita d'uso, la compatibilità con l'esecuzione parallela e la capacità degli ambienti di esecuzione di supportare la simulazione interattiva. Questo simulatore fornisce, per lo sviluppo ed il debugging delle topologie da simulare, un ambiente grafico molto intuitivo (GNED) contenente una rappresentazione testuale della topologia del modello, un'interfaccia grafica (Tkenv) per l'esecuzione della simulazione mediante la quale è possibile visionare i moduli del modello e vedere l'evolvere dello stato della simulazione, un'interfaccia da linea di comando (Cmdenv) ed infine due tool grafici (PLOVE e SCALARS) per visualizzare il contenuto degli output della simulazione scritti su appositi file dati generati automaticamente, ma estendibili dall'utente.
Oltre a questa serie di caratteristiche per la simulazione, OMNeT++ è dotato di un buon algoritmo di debugging, utile a rilevare gli errori simulando la rete passo passo. Il suo livello di astrazione è molto elevato e le sue prestazioni durante l'esecuzione sono di livello eccellente. La sua area di applicazione primaria, come detto prima, è la simulazione di network di comunicazione, ma grazie alla sua

architettura generica e flessibile è usata con successo in altre aree tra cui:

- Modellazione di protocolli
- Modellazione di code nelle reti
- Modellazione di multiprocessori e altri sistemi hardware distribuiti
- Convalida di architetture hardware
- Valutazione degli aspetti performanti dei sistemi software complessi
- Modellazione e simulazione di sistemi ad eventi discreti con modalità di comunicazione basata su scambio di messaggi

3.2.1 MODULI

Come abbiamo più volte ripetuto precedentemente, un modello implementato in OMNeT++ è costituito da una gerarchia di moduli annidati[18], che comunicano tra loro attraverso lo scambio di messaggi. I simple modules sono moduli attivi contenenti al loro interno gli algoritmi del modello, i quali vengono implementati dall'utente in C++ e sono realizzati come sottoclassi di cSimpleModule aventi funzioni membro virtuali ridefinite per definire le caratteristiche.

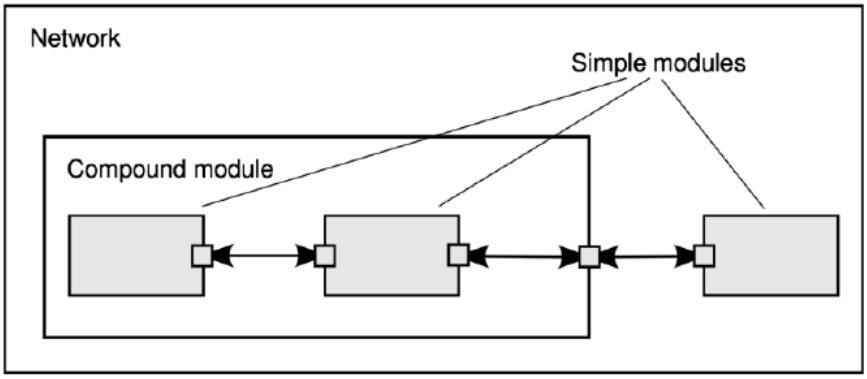

Figura 9: Descrizione architettura basata su Module

Più simple modules possono essere raggruppati insieme per formare compound modules (moduli composti), in cui il numero di livelli gerarchici è illimitato. L'intero modello (Figure 9), chiamato network (NED) in OMNeT++, è esso stesso un compound module le cui caratteristiche possono essere configurate dall'utente tramite implementazione.
I simple module sono i componenti attivi nel modello, in quanto al loro interno accadono gli eventi. Essi creano, inviano, ricevono, memorizzano, modificano, schedulano e distruggono i messaggi, usando le OMNeT++ class library come riferimento. Essi sviluppano di base le seguenti funzioni membro[18]

- Void initialize()
- Void handleMessage(cMessage *msg)
- Void activity()
- Void finish()

Nella fase di inizializzazione (funzione *initialize()*) si costruisce la rete, si creano i simple e compound module, i quali vengono connessi tra loro e configurati in base alle definizioni del file NED e del file di configurazione omnetpp.ini. Quest'ultimo è dotato di una specifica sintassi, grazie alla quale è possibile modificare i valori dei parametri senza dover ricompilare il modello, essendo tale file letto in fase di execute della simulazione. Durante il processamento degli eventi sono chiamate le funzioni *handleMessage()* (evocata da ogni messaggio che arriva al module, oppure in seguito a schedulazione ad un fissato istante di tempo) e *activity(),* la quale non necessita della preventiva esecuzione della funzione *initialize()*, ma presenta limiti notevoli dal punto di vista della scalabilità. All'interno di queste due funzioni l'utente, tramite implementazione ha la possibilità di personalizzare il comportamento del sistema, modificando le caratteristiche del modello. Solo nella fase finale è possibile memorizzare i dati statistici raccolti durante la simulazione (durante l'invocazione della funzione *finish()*).

3.2.2 MESSAGGI, INTERFACCIE e CONNESSIONI

Gli eventi, in Omnet++, sono rappresentati mediante messaggi, a partire da un'istanza della classe cMessage. Grazie alle interfaccie di

Input e Output, tra i vari moduli avviene lo scambio di tali messaggi, il quale può avvenire (Figure 10) o tra l'interfaccia di un sub-module e un compound module(caso b), o tra sub-module(caso a).

(a) Connessione tra sotto-moduli (b) Connessione tra livelli gerarchici

Figura 10: Connessione tra Module

Gli eventi in ingresso al FES (Future Event Set) sono consumati in base all'ordine d'arrivo dove, dati due messaggi, quello con il tempo d'arrivo più basso viene eseguito prima; se tali valori sono uguali, si esegue prima quello con il valore di priorità più basso e se anche le priorità coincidono si valuta quello schedulato/inviato prima. L'oggetto cMessage può rappresentare diverse entità tra cui messaggi, eventi, pacchetti, frame, celle, job, bit o segnali che viaggiano attraverso la rete. Tale oggetto ha degli attributi, usati dal simulation kernel, e dal programma di simulazione. Tra i più importanti ricordiamo: nome del messaggio, tipo, lunghezza, bit error flag (settato ad uno quando la probabilità d'errore supera una certa soglia), priorità (usata dal kernel per ordinare i messaggi in ingresso al FES), timestamp, informazioni di controllo, indicatore di contesto, attributi per la memorizzazione di informazioni come moduli, porte sorgente e destinazione, tempo di invio e di ricezione, ecc.
E' utile infine sapere, che è possibile estendere ogni message al fine di aggiungervi nuovi campi creando file con estensione *.msg, dal quale in seguito a compilazione, è generato codice C++ che fornisce i metodi all'utilizzatore per settare i nuovi campi definiti dall'utente.

3.2.3 PROGRAMMARE GLI ALGORITMI

I simple module, come detto, contengono al loro interno gli algoritmi che vengono implementati come funzioni in C++. Chi scrive la simulazione può usare tutta la potenza e la piena flessibilità di tale linguaggio di programmazione, supportata dalla libreria di classe di simulazione OMNeT++, tra le cui caratteristiche troviamo:

- Moduli, porte, parametri, canali
- Messaggi
- Pacchetti
- Classi di container (code, array, etc)
- Classi di raccolta dati
- Classi di stima e distribuzione statistica

Grazie a questa libreria di classi è possibile implementare i simple module in base alle necessita dell'utente. Compito poi dello sviluppatore, implementare nuove funzionalità da aggiungere a quelle presenti, così da modellarne il comportamento a proprio piacimento. Quello che occorre fare, in sostanza, non è altro che creare un nuovo progetto che si baserà su librerie pre-esistenti e modificarne le classi al loro interno, fino ad ottenere il risultato voluto. Come vedremo nel proseguo dell'elaborato di tesi, questa carattestica sarà fondamentale e alla base della buona riuscita del nostro progetto.

3.2.4 ARCHITETTURA

L'architettura di OMNeT++ è realizzata mediante i seguenti componenti[18] :

- Sim: è il simulation kernel ed una libreria di classi che l'utente può collegare alla propria simulazione. Questo componente gestisce gli eventi futuri, istanzia simple module ed altri componenti all'inizio dell'esecuzione della simulazione
- Envir: è una libreria contenente il codice comune a tutte le interfacce utente, consente di estendere le interfacce di base, mediante plug-in aggiuntivi. Esso mantiene il controllo di cosa accade durante la simulazione. Il metodo main() è localizzato in questo componente e determina quali modelli richiamare durante la simulazione
- Cmdenv e Tkenv: come detto, questi componenti sono le implementazioni di interfacce utente specifiche da collegare alla propria simulazione: la prima agisce da linea di comando, la seconda è grafica. Mediante queste interfacce l'utente ha la possibilità di iniziare e terminare la simulazione, nonchè seguire l'evolvere della simulazione

- Model Component Library: consiste in una serie di definizioni di simple module, tipi di compound module, canali, reti, messaggi ed altre entità legate alla simulazione
- Executing Model: è il modello che è stato impostato per la simulazione. Esso contiene oggetti (moduli, canali) che sono tutte istanze di componenti appartenenti al Model Component Library

Tutti questi componenti sono collegati tra loro mediante una logica di connessioni (Figure 11) che possiamo descrivere nel seguente modo:

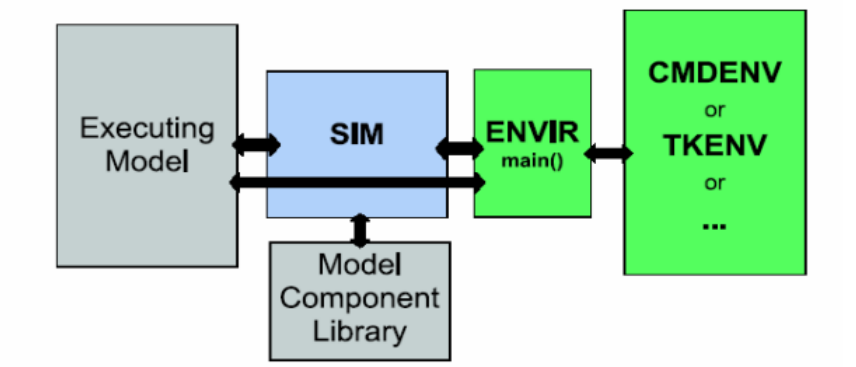

Figura 11: Architettura Omnet++

- Sim ↔ Executing Model: Sim invoca i moduli nell'Executing Model come eventi accaduti, memorizzandoli nell'oggetto principale di Sim ovvero la Simulation (della classe cSimulation, che memorizza i moduli del modello in esecuzione e tiene traccia degli eventi nella FES). Viceversa Executing Model chiama le funzioni contenute in Sim ed utilizza le classi di libreria ad essa associate
- Sim ↔ Model Component Library: Sim fa riferimento al Model Component Library quando durante la creazione dinamica dei moduli (fase di inizializzazione della simulazione)
- Executing Model ↔ Envir: l'oggetto EV (unica istanza della classe cEnvir, utilizzata per gestire le funzionalità dell'I/O e

fornire metodi per accedere a Sim), è usato per visualizzare informazioni durante la simulazione e per la scrittura sul file di log

- Sim $\leftrightarrow$ Envir: Envir invoca Sim per le funzionalità legate allo scheduling e all'esecuzione degli eventi, mentre Sim sfrutta Envir per produrre i risultati della simulazione
- Envir $\leftrightarrow$ Tkenv e Cmdenv: Envir definisce una classe per l'interfaccia utente chiamata TOmnetApp, le cui sottoclassi sono Tkenv e Cmdenv. E' l'utente poi, a selezionare quale delle due tipologie utilizzare a seconda delle necessità. Viceversa Tkenv e Cmdenv sfruttano alcune funzionalità di base che Envir offre.

Dopo questa descrizione architetturale, nel prossimo paragrafo spiegheremo come installare e configurare Omnet++ così da poterlo utilizzare nel proseguo della tesi.

3.3 INSTALLAZIONE E CONFIGURAZIONE DI OMNET++

In questo paragrafo vedremo come installare e configurare OMNET++[18]. Tutto ciò può sembrare banale e scontato, ma durante lo sviluppo del progetto alla base di questo elaborato, si è rivelato tutt'altro che semplice. Passiamo ora alla descrizione passo passo del processo di installazione e configurazione vera e propria di OMNeT++. I sistemi operativi supportati e descritti sono:

- Windows 7, Vista, XP
- Mac OS X 10.5 e 10.6
- Linux

Windows

1. Per eseguire OMNeT++ è necessario installare Java runtime (JRE). Scaricare, quindi, Java 5.0 o successivi dal sito http://www.java.com ed installarlo prima di procedere
2. Scaricare, quindi OMNeT++ dal sito http://omnetpp.org. Essere sicuri di scaricare il file seguente: omnetpp-4.2.2- win32.zip
3. Estrarre il file zip. Per fare ciò, click destro del mouse sul file zip e scegliere Estrai tutto dal menù. Si possono utilizzare anche altri programmi per l'estrazione del file (Winzip, 7zip, etc)
4. Controllare che, all'interno della cartella del file estratto, ci siano le cartelle doc, images, include, msys, etc e i file mingwenv.cmd, configure, Makefile e altri
5. Doppio click su mingwenv.cmd . Inserire nella finestra che si aprirà il seguente comando:
   $./configure
   $ make
   In questo modo il programma verrà installato sul computer
6. Al termine dell'installazione, digitare sempre su tale finestra il comando
   $ omnetpp
   OMNeT++ verrà quindi avviato

Mac Os X

Sono supportate le release Mac OS X 10.5 (Leopard) e Mac OS X 10.6 (Snow Leopard), testate sull'architettura Intel 32-bit.

1. Per eseguire OMNeT++ è necessario installare Xcode. Scaricare Xcode 4.0 dal sito http://developer.apple.com/technology/xcode.html
2. Scaricare, quindi OMNeT++ dal sito http://omnetpp.org. Essere sicuri di scaricare il file seguente: omnetpp-4.2-src.tgz
3. Aprire un terminale ed estrarre la cartella usando il comando seguente:
   $ tar zxvf omnetpp-4.2-src.tgz
   Verrà cosi creata una cartella chiamata omnetpp-4.2, contenente i file di simulazione
4. Sempre sul terminale digitare:
   $ ./configure
   $ make
   In questo modo il programma verrà installato sul computer
5. Una volta terminata l'installazione sarà possibile avviare il programma digitando:
   $ omnetpp

Linux

1. OMNeT++ richiede che vengano installati alcuni pacchetti sul computer. Questi pacchetti includono il compilatore C++ (gcc), Java runtime, e altre librerie e programmi
2. Scaricare, quindi OMNeT++ dal sito http://omnetpp.org. Essere sicuri di scaricare il file seguente: omnetpp-4.2-src.tgz
3. Aprire un terminale ed estrarre la cartella usando il comando seguente:
   $ tar zxvf omnetpp-4.2-src.tgz
   Verrà cosi creata una cartella chiamata omnetpp-4.2, contenente i file di simulazione
4. Sempre sul terminale digitare:
   $ ./configure
   $ make
   In questo modo il programma verrà installato sul computer

5. Una volta terminata l'installazione sarà possibile avviare il programma digitando:
   $ omnetpp

Dopo aver installato e configurato OMNET++ siamo pronti a passare alla parte tecnica e implementativa del progetto.
I prossimi capitoli costituiscono la parte operativa vera e propria del nostro elaborato di tesi. Al loro interno spiegheremo le finalità di questo progetto, il modello da simulare, il network utilizzato, gli algoritmi, i risultati ottenuti e le conclusioni che caratterizzano questa tesi.

4 SCENARIO DI BASE

In questo capitolo descriveremo lo scenario alla base del nostro progetto di laurea, ovvero il modello, raffigurato da un Network (NED) OMNET++, su cui verte la nostra analisi simulativa. Nella prima parte, analizzeremo i singoli componenti o moduli del NED, argomentando la nostra scelta e descrivendone funzionalità e implementazione. Nella seconda parte invece, ci focalizzeremo sull'analisi delle politiche di scheduling esistenti e degli algoritmi da noi realizzati, analizzandone pregi e difetti.
Prima di tutto questo però, introduciamo l' argomento, fornendo alcune nozioni di base alla teoria delle code che verranno utilizzatwe nel proseguo dell'elaborato di tesi.

4.1 INTRODUZIONE

Prima di passare alla descrizione del progetto è utile spiegare alcuni parametri raffiguranti le variabili quantificatrici che incontreremo durante l'elaborato, partendo da un modello che possiamo raffigurare in maniera astratta nel seguente modo (Figure 12)

Figura 12: Sistema Astratto del modello Job based

Tali variabili quantificatrici sono:

- T : la lunghezza del tempo totale di osservazione del sistema
- A: numero di arrival nel sistema
- F: numero di completation del sistema

Partendo da queste quantità è possibile definire:

1. λ, Arrival Rate:$\lambda = \frac{A}{T}$. Se ad esempio si verifica un numero di arrivi A pari a 8 e il tempo totale di osservazione T è 4 secondi allora $\lambda = \frac{8}{4} = 2\frac{richieste}{secondo}$. Da questo parametro è possibile calcolare anche l' Arrival Time ovvero il tempo che intercorre tra l'arrivo di una richiesta e l'arrivo della successiva, semplicemente calcolandone l'inverso.

2. X, Throughput: $X = \frac{F}{T}$. Questo parametro identifica il numero di richieste evase dal sistema rispetto al tempo di osservazione. Se ad esempio F=4 richieste e T =2 secondi allora $X = 4/2 = 2\ \frac{richieste}{secondo}$

3. B, Busy Time. Rappresenta il tempo in cui una risorsa è occupata

4. U,Utilization: $U = \frac{B}{T}$ Rapprenta quanto la percentale di tempo in cui una risorsa è occupata, rispetto al tempo totale di osservazione del sistema

Nel nostro progetto poi abbiamo utilizzato anche alcune variabili derivate più specificatamente dalla Queueing theory ovvero:

- SERVICE TIME: identifica il tempo in cui una determinata risorsa/componente riesce a completare una richiesta in input.
- QUEUE TIME: identifica il tempo che ogni richiesta trascorre in coda, in attesa di essere eseguita dal componente.
- RESPONSE TIME: identifica il tempo trascorso dalla richiesta nel sistema, dalla sua creazione al momento della sua terminazione

Dopo aver dato la definizione di queste caratteristiche possiamo passare alla descrizione vera e propria del nostro progetto di laurea.

4.2 NED Tesi Project

L'obiettivo del nostro progetto, come detto nella prefazione, è quello di trovare uno o più algoritmi di scheduling class-based che permettano di gestire al meglio il problema del Load Balancing all'interno di un sistema, basato su Cloud Compunting. Partendo dal modello descritto in "A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment" (2010) [19] noi sappiamo che ogni sistema Cloud, è composto da un insieme di server fisici collegati tra loro e formanti, nel loro insieme, un cluster. Ognuno di essi poi, attraverso il processo di virtualizzazione, crea un numero n di macchine virtuali per l'elaborazione delle richieste in ingresso. Ognuna di queste richieste, che noi rappresenteremo attraverso un elemento chiamato Job, deve essere associata ad una delle n macchine virtuali sopra citate, in modo da garantirgli il miglior/minor Service Time possibile. Abbiamo deciso di realizzare un modello in OMNET++, chiamato NED Tesi Project, che permetta di simulare il comportamenteo di uno di questi server fisici, al fine di studiarne le prestazioni attraverso alcuni parametri generati in output dal modello. Questo modello affonda le proprie radici nella Queueing Theory, grazie all'utilizzo di diversi elementi raffigurati da code e caratterizzati da un tempo di attesa(QueueTime) per l'erogazione del servizio proprio come, ad esempio, alla cassa di un supermarket.
Nel corso di questo paragrafo descriveremo nel dettaglio tale modello e tutti i suoi componenti.

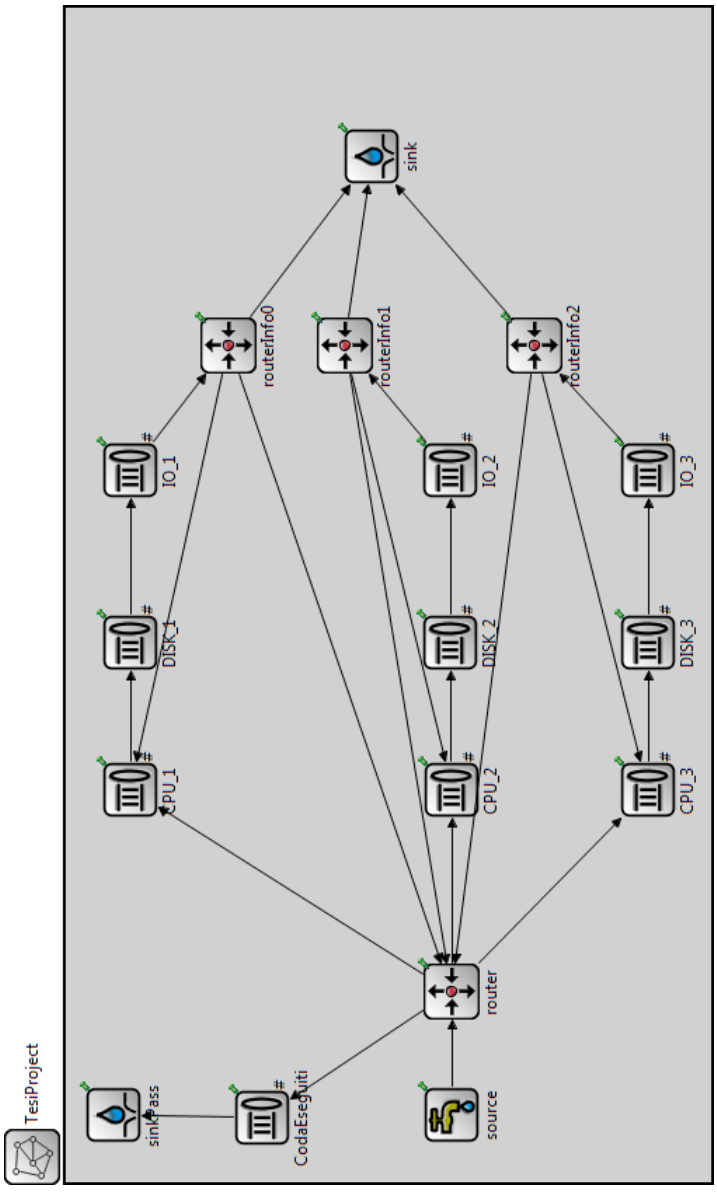

Figura 13: NED Tesi Project

La Figure 13 rappresenta una descrizione grafica di tale modello, che ci permette di mettere in luce tutti i componenti che lo compongono.
Come detto, l'obiettivo di questo NED è quello di simulare il com-

portamento di una macchina fisica, suddivisa in un numero prestabilito di macchine virtuali (nel nostro caso tre), a fronte dell'arrivo in Input di varie richieste di servizio. Noi sappiamo che secondo lo studio realizzato da Cicotti(2005) [20], vi sono quattro indicatori di performance per ogni macchina virtuale i-esima, vale a dire: CPU, memoria (MEM), dispositivi di memorizzazione (I / O) e la larghezza della banda di rete. Quest'ultimo parametro non verrà considerato nel proseguo della tesi in quanto, tra le ipotesi iniziali, abbiamo deciso di considerare i collegamenti tra i vari moduli perfettamente funzionanti e le varie capacità trascurabili al fine del calcolo delle prestazioni.
Come si può vedere ogni macchina virtuale i-esima è rappresentata dalla serie di tre code(moduli) aventi capacità infinita e identificate da un comportamento FIFO:

- CPU_m
- $DISK_m$
- IO_m

dove $m \epsilon M$ rappresenta le varie macchine virtuali del sistema.
Come vedremo, questi tre componenti saranno personalizzabili dallo user attraverso alcuni parametri che ne caratterizzeranno le prestazioni. A loro volta anche le richieste in ingresso (da ora in avanti chiamate Job) sono suddivise in varie classi. Il componente che genera i Job è chiamato Source, mentre il Sink è il modulo, all'interno del sistema,che permette la raccolta statistica dei dati in Output così da poter valutare le prestazioni degli algoritmi di routing implementati. La suddivisione (routing) delle richieste tra le macchine virtuali è gestita dal componente "pensante" del modello, ovvero il Router.
Per la realizzazione di tutti questi moduli siamo partiti, come detto nel capitolo precedente, da classi esistenti nella libreria Omnet++, estendendole e modificandone il codice (realizzato in C++) al loro interno al fine di ottenere il comportamento desiderato. La libreria di riferimento utilizzata nel nostro progetto, è chiamata "queueinglib" ed è di riferimento per tutti i progetti di simulazione di Network in Omnet++. Quello che troveremo all'interno del progetto, sarà quindi una serie di moduli/componenti formati essenzialmente da tre file:

1. File NED: permette la dichiarazione di alcune variabili che possono essere utilizzate all'interno di TUTTI i metodi della classe. Al suo interno sono presenti inoltre le caratteristiche grafiche del modulo, a livello di interfaccia.
2. File .cc: è il file che contiene l'implementazione C++ della classe e dei suoi metodi
3. File .h: questo file contiente la dichiarazione dei metodi, i costruttori e i distruttori della classe

Dopo questa breve sintesi raffigurante la totalità del progetto e del modulo alla base, nei prossimi paragrafi andremo ad analizzare singolarmente i vari componenti così da avere una visione più chiara e precisa della composizione del nostro progetto.

4.2.1 FILE INI

Il primo oggetto che vediamo, non è un componente vero e proprio, ma bensì un file essenziale per qualsiasi progetto Omnet++. Attraverso questo documento è possibile settare, sia tutte le variabili dichiarate nelle classi del project, sia le variabili cosiddette di simulazione. Per fare ciò Omnet++, mette a disposizione un'interfaccia grafica molto semplice ed intuitiva che permette di selezionare un qualsiasi parametro e assegnarvi un valore predefinito, ed in alternativa un'interfaccia testuale che svolge le medesime mansioni. I passi necessari alla creazione di un file .INI da IDE Eclipse, sono i seguenti:

1. File
2. New
3. Initialization File
4. Si attribuisce un nome univoco al file INI e lo si associa al progetto di riferimento
5. Si termina la creazione con il pulsante Finish

Quello che otterremo è la seguente schermata:

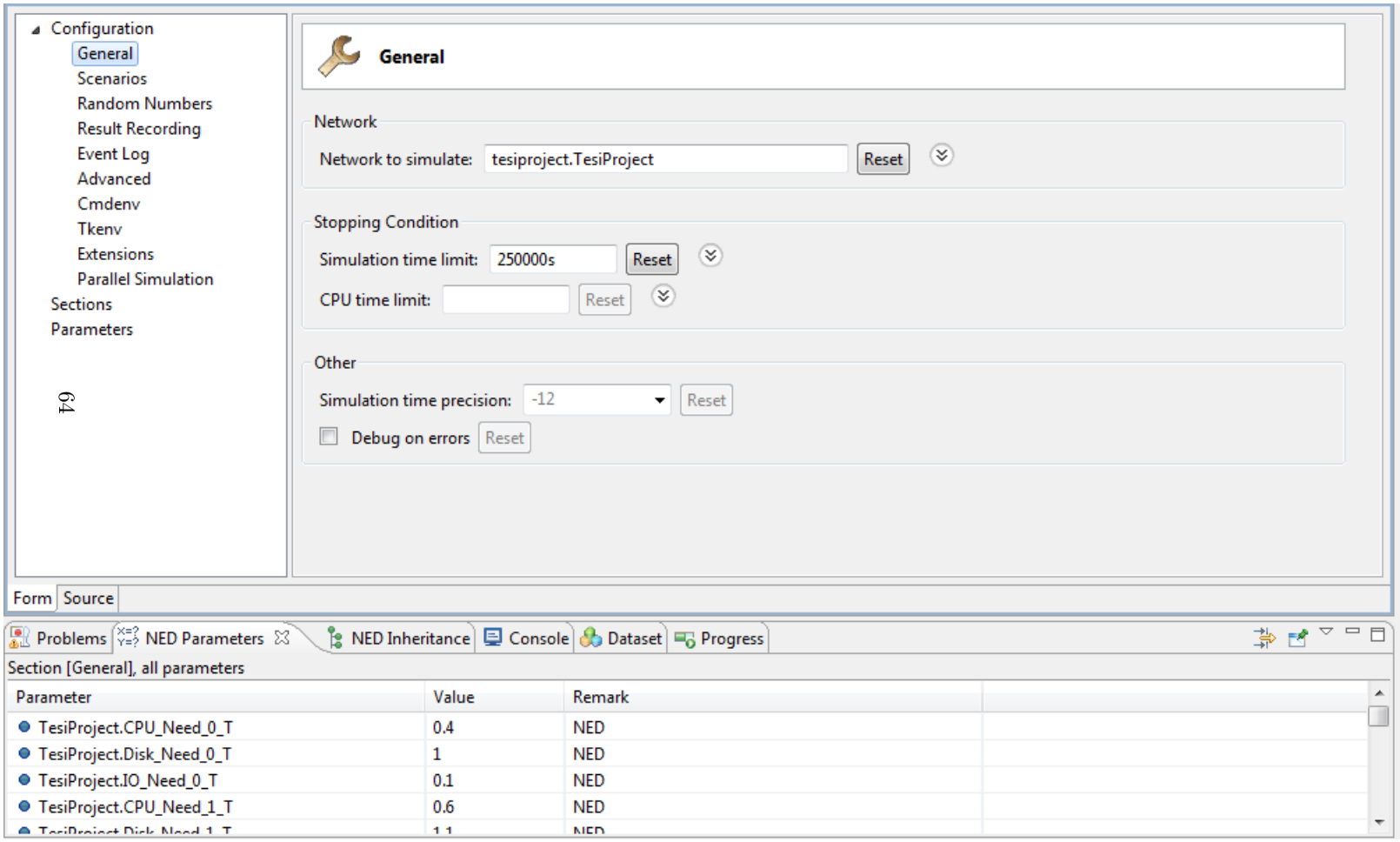

Figura 14: File INI

Come vediamo, nella sezione di sinistra, sono posti tutti i sotto menù delle parti del progetto su cui si vorranno apportare modifiche. La parte centrale è lo sviluppo di tali menù, dove per esempio è possibile (come da Figure 14) settare il tempo di simulazione. Nella parte inferiore invece, è presente un elenco di tutte le variabili dichiarate nei vari file NED delle classi. Attraverso questo menù è possibile collegarsi direttamente a tale dichiarazione, ed istanziarvi un valore a nostro piacimento.
I parametri più importanti (nel nostro progetto) da modificare nel file INI sono:

- SIMULATION TIME LIMIT (General): indica il tempo di simulazione di ogni singola run
- REPEAT COUNT (Scenarios): indica quante ripetizioni, per ogni configurazione di simulazione, vengono eseguite
- ENABLE REC OF VECTOR (Result Recording): se settato a TRUE permette di memorizzare i risultati statistici della simulazione, rendendone così immediata la lettura
- ENABLE EVENT-LOG REC (Event Log): se settato a TRUE permette di creare un file log che contiene tutti i valori, con gli annessi istanti di tempo, delle variabili di cui si vuole studiare l'evoluzione nel tempo.
- Parameters: questo intero sottomenù permette l'assegnamento di valori alle varie variabili di progetto

Una volta creato il file INI, come vedremo, sarà possibile associarlo ad una Run Configuration così da creare una simulazione che utilizzi tutte le modifiche da noi apportate e generi dei risultati prestazionali in Output

4.2.2 MACCHINE VIRTUALI

Tornando ai componenti veri e propri, come detto, la nostra macchina fisica è suddivisa in tre virtual machine, ognuna delle quali è composta da tre code distinte che simulano il comportamento di uno specifico componente. Il primo che si incontra è la CPU, il secondo il DISK e il terzo l'I/O. Questa ovviamente è una semplificazione del modello reale ma risulta comunque essere un buon compromesso[20] in termini di qualità e veridicità di simulazione.

Ogni macchina virtuale è caratterizzata da tre valori, uno per ogni componente interno, i quali identificano il livello prestazionale associato rispettivamente a CPU, DISK e I/O. Questi parametri risulteranno fondamentali in seguito per l'analisi delle prestazioni del modello. L'uscita della serie dei tre suddetti componenti è collegata ad un router particolare, chiamato RouterInfo, il quale ha sia la funzione di inviare informazioni al Router principale, sia la funzione di inoltrare i JOB che hanno terminato la loro esecuzione al Sink in modo da analizzarne i valori prestazionali.
Per capire come funziona nel dettaglio ogni componente, intervalleremo concetti prettamente teorici a pseudo-codice così da rendere più chiara la spiegazione. Il primo componente che analizziamo nel dettaglio è la Coda. Nel nostro trattato di tesi abbiamo deciso di introdurre un'ipotesi iniziale molto forte riguardo questo componente, come detto, ossia porre la capacità uguale a infinito, così da non dover gestire il problema del Queue overflow. Tornando alla descrizione del nostro progetto, a livello implementativo, per la realizzazione delle code siamo partiti da un CSimpleModule presente nella libreria sopra citata, chiamato Queue e abbiamo così creato tre classi da esso derivate chiamate rispettivamente:

- QueueCPU
- QueueDisk
- QueueIO

Il comportamente di questi tre componenti è il medesimo, indifferentemente dal tipo, ossia quando un job si presenta in Input, si scatena il metodo handleMessage(cMessage *msg), il quale aggiorna i valori di timing del Job e la Event List del sistema e nel caso in cui la coda sia piena il Job viene scartato, altrimenti viene inoltrato al componente successivo. L'aspetto che differenzia le tre tipologie di componente sono i parametri che le identificano. Per ognuna di queste classi esistono un numero n di valori (con n pari al numero di macchine virtuali, nel nostro caso tre) che rappresentano la capacità prestazionale del singolo componente k della macchina virtuale m, chiamati $\alpha_{k,m}$. Tanto più alfa è piccolo tanto più sarà minore il Service Time associato al componente k, in quanto legati da proporzionalità diretta. Un esempio numerico può essere utile a rendere più chiaro questo concetto. Supponiamo che $\alpha_{CPU,VM1} = 1$(dove

$\alpha_{CPU,VM1}$ raffigura la capacità prestazionale della CPU della Virtual Machine uno) e $\alpha_{CPU,VM2} = 1.5$ allora, come vedremo in seguito, il ServiceTime della CPU della macchina uno (a parità dei restanti parametri) sarà minore del ServiceTime della CPU della macchina due. Esistono quindi in totale, nel nostro progetto, nove valori di alfa che devono essere settati dall'utente del simulatore all'interno del file INI affinchè la simulazione possa iniziare.
Il calcolo del Service Time spetta proprio alle varie Queue e diventa quindi indispensabile, per il proseguo dell'elaborato, capire come viene calcolato numericamente questo parametro nel nostro modello. Per fare ciò è necessario passare al paragrafo successivo che spiegherà nel dettaglio il comportamento e le caratteristiche di un altro elemento chiave del nostro modello ovvero il JOB.

4.2.3 JOB

Tutte le richeste di servizio in ingresso al sistema possono essere scomposte in gruppi di componenti più piccoli, chiamati Job. Per esempio, un'applicazione che elabora un'immagine molto grande, può scomporre questo file in parti più piccole cosicchè possano essere elaborate in parallelo su più postazioni di lavoro distinte. In informatica, la sequenza logica di Job di una determinata richiesta di servizio è chiamato flusso di lavoro, o Workflow[21]. Nel nostro progetto, l'elemento Job rappresenta le richieste in input al sistema di simulazione che circolano all'interno della Network. Esso implementativamente, non è altro che un estensione della classe CMessage, della quale siamo andati a sfrutttare diversi parametri che si sono rivelati molto utili durante la realizzazione del modello. Tra questi ricordiamo:

- ID: permette di identificare univocamente ogni Job
- Kind: rappresenta la tipologia del Job e grazie ai metodi getKind() e setKind() è stato così possibile, dividere le richieste in ingresso in varie classi (nel nostro caso tre:A,B,C)
- getSenderModuleID: permette di identificare univocamente il module dal qual è arrivato il nostro Job
- getArrivalModuleID: permette di identificare univocamente il module al quale è arrivato il nostro Job

- Priority: rappresenta la priorità del nostro module
- TotalQueueingTime: rappresenta il tempo che ogni Job spende in coda durante il tempo in cui è attivo. Grazie ai due metodi getTotalQueueingTime() e setTotalQueuingTime() è così possibile ottenere informazioni riguardo allo stato del sistema e delle singole macchine virtuali ed inoltre ottenere dati statistici sul tempo impiegato a servire un determinato Job
- TotalServiceTime: rappresenta il tempo che ogni Job spende ad essere servito da una delle code del sistema. Grazie a questa informazione e alla precedente(QueueingTime) è possibile ricavare il ResponseTime ovvero, come detto, il tempo totale trascorso dal Job nel sistema, dal momento del suo arrivo alla fine della sua computazione.

Al fine di rendere il nostro modello più simile alla realtà, i vari Job sono suddivisi in classi secondo il Multiple Class Model. Questo modello consiste in un numero C di classi ognuna dell quali, come vedremo, sarà caratterizzata da diversi parametri che la identificano univocamente rispetto alle altre. Questo comportamento renderà gli algoritmi che implementeremo di tipo Class-Based.
Ogni Job nell'istante della sua creazione, con una determinata probabilità, viene associato ad una classe (come vedremo nel paragrafo relativo al componente Source) e così facendo ne eredità i parametri fondamentali. Questi parametri, sono essenzialmente due e si possono riassumere nel seguente modo:

1. Coeffinciente Classe-Componente $S_{c,k}$
2. Average Number of Round $AvNumberRound_c$

Il primo viene utilizzato per il calcolo del Service Time di ogni componente k delle singole virtual machine e varia a seconda della classe della richiesta in Input e a seconda del componente, mentre il secondo parametro permette di calcolare la probabilità che un determinato Job di classe c richieda un'elaborazione multipla dal sistema.
Partendo dal primo parametro, chiamiamo $S_{c,k}$ il tempo che impiega il componente k per la singola elaborazione di un Job di classe c, sotto la condizione di *pari capacità computazionali dei componenti* che vedremo nel proseguo della tesi. Questa ultima nota tiene in

considerazione il fatto che il calcolo del Service Time, non dipende esclusivamente da $S_{c,k}$, ma anche dalle caratteristiche prestazionali dei singoli componenti della virtual machine e dal concetto di Average Number of Round che vedremo tra poco. Il nostro progetto, come detto, è formato da tre classi c di Job e tre componenti k, grazie alle quali è possibile costruire la seguente matrice raffigurante i vari $S_{c,k}$:

$$\begin{array}{cccc} & CPU & DISK & IO \\ JOB_A & S_{A,CPU} & S_{A,DISK} & S_{A,IO} \\ JOB_B & S_{B,CPU} & S_{B,DISK} & S_{B,IO} \\ JOB_C & S_{C,CPU} & S_{C,DISK} & S_{C,IO} \end{array}$$

Sarà compito dell'utente, quando avvia la simulazione o attraverso la modifica del file INI, impostare i valori di tali parametri a proprio piacimento, così da rappresentare al meglio la realtà simulata.
Il secondo parametro identificativo delle varie classi di Job è l'Average Number of Round. Questo parametro identifica il numero di elaborazioni necessarie al Job, appartente alla specifica classe, a terminare la sua esecuzione. In altre parole possiamo dire che la variabile $AvNumberRound_c$ identifica il numero di round che il Job effettua all'interno del seguente sistema (Figure 15):

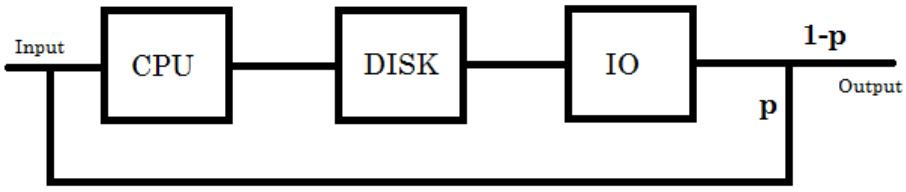

Figura 15: Parametro Round

Ogni Job al termine della sua elaborazione, non viene immediatamente inoltrato al sink, ma con probabilità p effettua un round e torna in Input alla virtual machine per essere elaborato nuovamente. Il parametro settato dall'utente identifica il numero medio di round che ogni Job, di una determinata classe c, deve eseguire pri-

ma di essere completato. Grazie a tale valore è possibile calcolare la probabilità p che avvenga tale Round come segue:

$$p = 1 - \frac{1}{AvNumberRound_c}$$

dove $AvNumberRound_c$ deve appartenere alla classe dei numeri interi. Per la realizzazione di questo meccanismo viene utilizzato, ancora una volta, il componente RouterInfo. Spieghiamo come attraverso poche righe di pseudo codice:

```
int NumCasuale=rand%100
if (ClasseJob==A){
if(NumCasuale>p_A) Job in Output
else Job ritorna in Input
}
//p_A rappresenta la probabilità della classe A
//moltiplicata per cento
```

Il metodo rand() attribuisce un valore random alla variabile "NumCasuale": se questo numero è maggiore della probabilità di ritorno (p_A) settata dall'utente, il Job termina la sua esecuzione e viene inoltrato al Sink, altrimenti, con probabilità $1 - p_A$ torna in Input alla CPU della macchina virtuale così da poter essere rielaborato.
Dopo aver definito questi primi due componenti del nostro NED project abbiamo i mezzi matematici e scientifici per calcolare numericamente il Service Time di ogni componente e del sistema in generale.

Service Time

Il service time come detto, identifica il tempo (espresso in secondi), in cui un determinato componente soddisfa la richiesta di elaborazione di un Job in Input. Per prima cosa è importante sapere che, all'interno del nostro progetto, questo parametro viene descritto attraverso una distribuzione esponenziale (chiamata anche di Poisson) di parametro τ [22]. In teoria delle probabilità, la distribuzione di Poisson (o poissoniana), è una distribuzione di probabilità discreta che esprime le probabilità per il numero di eventi che si verificano successivamente ed indipendentemente in un dato intervallo di tempo, sapendo che mediamente se ne verifica un numero τ. Una possibile applicazione di tale distribuzione può essere la misurazione del

numero di chiamate ricevute in un call-center in un determinato arco temporale, come una mattinata lavorativa. Questa distribuzione $P(\tau)$, è anche nota come legge degli eventi rari è così descritta:

$$P(n) = e^{-\tau}\frac{\tau^n}{n!}$$

per ogni $n \in N$, dove τ è il numero medio di eventi per intervallo di tempo, mentre n è il numero di eventi per intervallo di tempo(lo stesso col quale si misura τ) di cui si vuole la probabilità.
Quello che noi andiamo a calcolare numericamente, per ogni componente, è il parametro τ che ne identifica la distribuzione.
Questo parametro è dato dal prodotto del coefficiente $\alpha_{k,m}$, identificativo delle prestazioni del componente k di una determinata macchina virtuale m, per la variabile $AvNumberRound_c$ e per il parametro $S_{c,k}$ i cui significati sono stati spiegati precedentemente. Matematicamente:

$$\tau = \alpha_{k,m} * S_{c,k} * AvNumberRound_c$$

Nel caso in cui $\alpha_{k,m} = 1(\forall k \epsilon K, \forall m \epsilon M$ con $K = \{CPU, DISK, IO\}$e $M = \{VM1, VM2, VM3\})$, si verifica la condizione di *pari capacità computazionali dei componenti,* in cui il prodotto $S_{c,k} * AvNumberRound_c$ descrive esattamente il tempo che impiega il componente k a servire un Job di classe c.
Se per esempio $S_{A,CPU} = 1.2s$, $AvNumerRound_A = 3$ e $\alpha_{CPU,VM1} = 0.8$ allora il tempo trascorso in media, in elaborazione da un Job_A in CPU, è dato da $S_{A,CPU} * \alpha_{CPU,VM1} * AvNumberRound_A = 2.88s$
.

4.2.4 SOURCE

Un altro componente molto importante per la realizzazione del nostro progetto è il Source. Esso, attraverso una distribuzione Poissoniana di parametro β, genera una serie di richieste che rappresentano l'Input del nostro sistema, ovvero quello che viene tecnicamente detto Arrival Time. Queste richieste poi, con probabilità prefissata $p_{Job,c}$ (dove c identifica la classe), come detto, vengono suddivise nelle varie classi di Job (nel nostro caso tre). Per comprendere maggiormente il meccanismo di suddivisione delle richieste postiamo di

seguito uno spezzone di pseudo codice che si basa ancora una volta sulla generazione di un numero casuale NumRand:

```
int NumRand=rand%3;
switch (NumRand)
{
case(0):
job->setKind(0);
break;
case(1):
job->setKind(1);
break;
case(2):
job->setKind(2);
break;
}
```

A seconda del valore della variabile NumRand, il cursore accede ad un particolare ramo dello switch, in cui viene settata la tipologia di classe del Job(e la priorità). Nel caso descritto dalla porzione di codice di cui sopra, viene riassunto il caso in cui $p_{Job,A} = p_{Job,cB} = p_{Job,C} = \frac{1}{3}$. Se volessimo modificare tali parametri, sarebbe sufficiente cambiare la "variabile guida" dello Switch Case, NumRand, in modo che le probabilità di cadere in un ramo rispetto ad un altro siano differenti.
Ritornando ai valori del parametro β, è importante sapere che vengono prestabiliti nel file INI, ognuno per una determinata run della simulazione suddivise per algoritmo. Così facendo è possibile studiare, come vedremo, il behaviour del sistema al variare dei parametri in ingresso (nel nostro caso l'Arrival Time o dell'ArrivalRate). La scelta del valore del parametroβ è quindi essenziale per evitare che il sistema non cada in condizione di instabilità, ovvero che le code non si sovraccarichino fino all'overflow. Esistono due possibili casi di studio per le condizioni limite per la stabilità/instabilità del modello che vengono presentati nel libro "Quantitative System Performance. Computer System Analysis Using Queueing Network Models"(1984) [23] e che abbiamo deciso di esporre

- Modello Single Class: ovvero tutti le richieste in Input appartengono alla stessa classe
- Modello Multiple Class: i Job in ingresso vengono suddivisi tra n classi differenti

Nel nostro progetto il modello del sistema è class-based e quindi utilizzeremo i valori ricavati dal secondo dei due casi sopra citati, ma per completezza li riportiamo entrambi come segue:

Modello Single Class

Questo primo caso analizza il caso di modelli con una sola classe di Job, o con Job aventi caratteristiche pressochè identiche. Noi sappiamo che dato un Arrival Rate η e un Service Rate λ, dove $\eta = \frac{1}{\beta}$ e $\lambda = \frac{1}{\tau}$ (con βche rappresenta l'InterArrivalTime eτ il Service Time) la condizione affinché il sistema sia stabile è:
$\rho < 1 \Rightarrow \frac{\eta}{\lambda} < 1$
ovvero che l'Arrival Rate sia minore del ServiceRate. Ponendo il focus sui tempi invece:
$\frac{\tau}{\beta} < 1$
In linea teorica, così facendo, tutte le richieste in ingresso vengono soddisfatte prima dell'arrivo del Job successivo. La realtà però considera anche altri parametri che portano ad un espansione del response time e quindi alla creazione di ritardo sulla coda, come ad esempio la probabilità di Round del job.

Modello Multiple Class

Questo secondo caso rappresenta il nostro modello composto, come detto, da tre classi di Job. Supponendo di avere un vettore $\overrightarrow{\eta}$ di Arrival Rate, ovvero un determinato η per ogni classe di Job, si dice che il sistema è stabile se la seguente condizione di stabilità è soddisfatta:

$$max_k \left\{ \sum_{c=1}^{C} \eta_c D_{c,k} \right\} < 1$$

dove $c \in C$ indica la classe, $k \in K$ il singolo componente delle macchine virtuali e $D_{c,k}$ indica il Demand della Classe c riferito al componente k. Quest'ultimo parametro nel nostro caso viene calcolato come:
$D_{c.k} = AvNumberRound_c * S_{c,k}$
Definiamo ora anche il concetto di Utilization($U_{c,k}$) di una determinata risorsa k da parte di una classe di Job c. Questo parametro nuemricamente può anceh essere calcolato come$U_{c,k} = \eta_c D_{c,k}$

da cui possiamo derivare una seconda versione della condizione di stabilità ovvero:

$$max_k \left\{ \sum_{c=1}^{C} U_{c,k} \right\} < 1$$

Nel nostro caso gli Arrivale Rate delle varie classi sono derivati dall'Arrival Rate del sistema ovvero:

$$\eta_c = \eta p_{Job,c}$$

dove $p_{Job,c}$ è la probabilità di un determinato Job di appartenere alla classe c.

4.2.5 SINK

Il Sink è il componente chiave per la raccolta dei dati statistici del progetto. Grazie a questo componente è possibile ottenere informazioni riguardanti le prestazioni del sistema, allo scopo di valutare gli algoritmi di routing sviluppati e implementati all'interno del progetto.
Quando un Job termina l'esecuzione, come detto, viene mandato in Input al Sink il quale lo analizza e attraverso la lettura di alcuni parametri da noi configurati, lo trasforma in informazioni statisticamente rilevanti. Tre sono i parametri possibili, utilizzabili al fine di valutare la capacità prestazionale del sistema:

Average Response Time

Rappresenta il tempo medio trascorso dalle richieste (suddivise per classe) all'interno del sistema. Ad ogni classe di job viene attribuito un valore medio che contribuisce a confrontare il livello prestazionale dei vari algoritmi. La formula utilizzata per calcolare tale valore è semplice:

$$AvResponseTime_{ClasseJob} = \frac{TotalResponseTime_{ClasseJob}}{NrJob_{ClasseJob}}$$

Per capire implementativamente come ottenere queste informazioni dalla successione dei Job in Input al Sink utilizziamo ancora una volta alcune righe di pseudo codice:

```
//All'arrivo di ogni Job si controlla la classe a cui
//appartiene(noi vediamo il caso di Job_Class=A)
//e si sviluppa la seguente istruzione condizionale
if(JobTipo==A)
{
NrJobA++;
ResponseTimeJobCorrente=job->getServiceTime()+job->getQueueTime();
ResponseTimeTotJobA=ResponseTimeTotJobA+ResponseTimeJobCorrente;
AverageResponseTimeJobA=ResponseTimeTotJobA/NrJobA;
}
```

Durante l'inizializzazione del Sink si creano:

- $NrJob_{ClasseJob}$ variabile contatore utilizzata per contare il numero di Job per classe terminati
- $TotalResponseTime_{ClasseJob}$ variabile (inizialmente posta a zero) utilizzata per la memorizzazione della somma dei ResponseTime appartenenti ai vari Job (suddivisi per classe)
- $AvResponseTime_{ClasseJob}$ variabile utilizzata per contenere la media dei response time

Average Queue Time

Rapprensenta il tempo medio speso dai Job(suddivisi per classe) in coda, in attesa del servizio. Ad ogni classe quindi è associato un determinato valore di AvQueueTime il cui valore è:

$$AvQueueTime_{ClasseJob} = \frac{TotalQueueTime_{ClasseJob}}{NrJob_{ClasseJob}}$$

Il codice utilizzato per implementarlo è simile al seguente:

```
//All'arrivo di ogni Job si controlla la classe a cui
//appartiene(noi vediamo il caso di Job Class=A)
//e si sviluppa la seguente istruzione condizionale
If(JobTipo==A)
{
NrJobA++;
QueueTimeCorrente=job->getQueueTime();
QueueTimeTotJobA=QueueTimeTotJobA+QueueTimeCorrente;
AvQueueTimeJobA=QueueTimeTotJobA/NrJobA;
}
```

Proprio come il caso precedente le variabili utilizzate vengono create e settate in fase di inizializzazione del Sink.

Deviazione Standard Response Time

Rappresenta la deviazione standard, riferita al Response Time calcolato sui singoli Job, suddivisi per classe. Questo parametro è molto utile al fine di costruire come vedremo un intervallo di confidenza per le caratteristiche degli algoritmi proposti.
Ricordiamo che la deviazione standard, o scarto quadratico medio, è calcolata come la radice quadrata della varianza che viene definita come il valore atteso del quadrato della variabile aleatoria centrata $x - E(x)$ ovvero:

$$\sigma^2(x) = E\left[(x - E\left[x\right])^2\right]$$

che spesso viene trasformata per comodità di calcolo in:

$$\sigma^2(x) = E\left[x^2\right] - E\left[x\right]^2$$

e proprio questo concetto abbiamo sviluppato grazie al seguente pseudo codice

```
//All'arrivo di ogni Job si controlla la classe a cui
//appartiene(noi vediamo il caso di Job Class=A)e
//si sviluppa la seguente istruzione condizionale
If(JobTipo==A)
{
NrJobA++;
RespTimeJobCorrente=job->getServiceTime()+job->getQueueTime();
RespTimeTotJobA=RespTimeTotJobA+RespTimeJobCorrente;
QuadRespTimeTotJobA=QuadRespTimeTotJobA+RespTimeJobCorrente^2;
AvRespTimeJobA=RespTimeTotJobA/NrJobA;
VarRespTimeJobA=(QuadRespTimeTotJobA/NrJobA)-AvRespTimeJobA^2;
DevStdRespTimeJobA=sqrt(VarianzaRespTimeJobA);
}
```

Quando calcoliamo la varianza utilizziamo la proprietà vista pocanzi che mi permette di utilizzare i due valori attesi calcolati poco prima. La trasformazione in Deviazione Standard, come si vede, consiste semplicemente nell'estrazione della radice quadrata della varianza. Proprio come il caso precedente le variabili utilizzate vengono create e settate in fase di inizializzazione del Sink.

Tra questi tre parametri, nel nostro progetto, abbiamo deciso di selezionare, come metro di giudizio delle prestazioni riferite ai vari algoritmi, l'Average Response Time.

Per essere analizzati però, i valori di tale parametro, devono essere trasformati in statistiche che compaiano al termine della simulazione. Per fare ciò è necessario utilizzare delle variabili particolari di tipo simsignal_t e una serie di passaggi tra parametri che portano all'emissione di tale informazione. Vediamo ora nel dettaglio, questo meccanismo di estrapolazione di dati, attraverso un segmento di codice della classe Sink.cc(l'esempio è basato sull'Average Response Time dei Job di classe A):

1. Nel file Sink.Ned per ogni variabile che si vuole tramutare in statistica occorre aggiungere la seguente dicitura:
   @signal[AvResponseTimeJobA](type="simtime_t");
   @statistic[AvResponseTimeJobA](title="Average ResponseTime Job class A";unit=s;record=vector,mean,max);

2. Nel file Sink.h
   simsignal_t AvResponseTimeJobASignal;

3. Nel file Sink.cc durante la fasi di inizializzazione (metodo initialize()) occorre legare le 2 variabili create rispettivamente nel file .h e .ned
   AvResponseTimeJobASignal=registerSignal("AvResponseTimeJobA");

4. Nel file Sink.cc, all'interno del metodo handleMessage(cMessage *msg), si effettuano le operazioni necessarie a calcolare il valore del parametro (nel nostro caso otterremmo il valore della variabile AvResponseTimeJobA) come descritto in precedenza

5. Nel file Sink.cc, all'interno del metodo finish() si emette tale valore affinche compaia nel file che racchiude le statistiche della simulazione nel seguente modo
   emit(AvResponseTime1Signal,RespMedio1);

Grazie a questi cinque passaggi, a fine simulazione, otterremo una serie di valori, raggruppati in tabelle, facilmente leggibili. Come creare il file che racchiude tali statistiche sarà un argomento trattato nei prossimi capitoli.

4.2.6 ROUTER

Il router come detto è il componente chiave del modello, suo è il compito di scegliere a seconda del valore di alcune caratteristiche,

quale sia il percorso migliore da associare al Job. Al suo interno infatti, questo componente contiene l'implementazione vera e propria di tutti gli algoritmi da noi trattati e, a seconda della scelta, il router agirà in maniera differente.
All'interno del file Router.cc, inizialmente vengono create le variabili cosidette "globali", ovvero che possono essere utilizzate in più parti della classa. Nel metodo *initialize()* vengono fatti controlli di correttezza sui dati in Input e viene associato l'algoritmo di routing desiderato. All'interno del codice del metodo *handleMessage(*msg)* è possibile visualizzare l'implementazione fisica di tali algoritmi con tutte le loro peculiarità.
E' importante ricordare come il router contenga al suo interno, una porzione di codice dedicata al riconoscimento della tipologia di Job in ingresso. Nel caso in cui essi non siano richieste di servizio ma messaggi di informazioni inviate dal componente RouterInfo, questi vengono letti e spediti in una Queue creata appositamente per il loro smaltimento.

4.2.7 FILE NED

L'ultimo componente che vediamo è il file NED nella sua totalità. Abbiamo deciso di dedicare una sezione a questo componente in quanto, oltre ad essere la rappresentazione vera e propria del modello alla base del progetto, al suo interno contiene tutti quei parametri, i cui valori vengono settati dall'utente. Il comportamento del file NED è semplice:

1. Legge i parametri in Input
2. Effettua dei controlli sulla loro correttezza e sull'accuratezza dei dati
3. Fornisce i valori dei parametri in input ai vari componenti che ne necessitano

In questo modo il router, pur essendo a valle di componenti come la CPU, il DISK ecc. conosce i parametri e i valori prestazionali che li contraddistinguono. Questo vedremo sarà utile, se non indispensabile, durante l'implementazione dei singoli algoritmi ma non solo. Si può quindi dire, che il NED oltre a contenere al suo interno tutti i componenti del modello funge anche da libreria di parametri e di indicatori di prestazione dei vari moduli.

4.3 ALGORITMI DI ROUTING

Come detto la principale funzione del router nel nostro modello, è quella a quale macchina virtuale indirizzare i Job in ingresso. In questo capitolo vedremo gli algoritmi di routing che vengono utilizzati in questo progetto, partendo da quelli già esistenti in letteratura e concludendo con quelli da noi appositamente creati. Di ognuno di essi descriveremo l'implementazione attraverso porzioni di pseudo codice. le caratteristiche e i principali pregi e difetti.
Prima di tutto ciò, introduciamo l'argomento con qualche nozione aggiuntiva riguardante il routing, per poi addentrarci nel caso specifico del nostro progetto.
Ogni router al suo interno possiede alcune tabelle di instradamento, che associano ad ogni possibile destinazione, la linea d'uscita da utilizzare. Questa linea di output è chiamata, nel nostro progetto, outGateIndex e assumerà valori da 0 a 2 suddivisi nel seguente modo:

- outGateIndex=0 se la destinazione è la macchina virtuale numero 1
- outGateIndex=1 se la destinazione è la macchina virtuale numero 2
- outGateIndex=2 se la destinazione è la macchina virtuale numero 3

Il router, grazie agli algoritmi di routing, ciascuno avente un impatto diverso sulla rete, decide quale percorso risulterà migliore per il Job in input. I requisiti che si desidera gli algoritmi di routing abbiano, nel nostro caso sono due e si possono riassumere nei seguenti criteri:

- Ottimalità: il Job viene mandato nella direzione che lo porta ad avere un Response Time minore possibile
- Semplicità: la soluzione non è la migliore ma è di facile realizzazione e in termini computazionali di calcolo poco dispendiosa

Tutto ciò che riguarda l'affidabilità e la robustezza delle connessioni non è considerato in questo trattato in quanto, viene assunta come condizione iniziale, il fatto che tutte le connessioni tra i componenti non possano subire guasti.
E' importante sapere inoltre che ogni algoritmo di routing può essere suddiviso in due grosse categorie ovvero:

1. ALGORITMI STATICI: in questo caso i criteri di scelta non si adattano ai cambiamenti della rete, ma bensì sono scelti in fase di inizializzazione
2. ALGORITMI DINAMICI: il router è in grado di cambiare il percorso in base alle condizioni del modello e quindi scegliere il percorso migliore

Dopo questa breve introduzione, andiamo a presentare gli algoritmi che verranno analizzati nel proseguo della tesi, ripartendoli in due grosse categorie secondo la suddivisione consigliata in "Analysis of Load Balancing Algorithms in P2P Streaming"(2008) [24].

- Randomized Algorithm: questa tipologia di algoritmo non considera in nessun modo il carico presente in rete, ma agisce in maniera del tutto indipendente. Fanno parte di questa categoria gli algoritmi Random e Round Robin
- Load Dependent Algorithm: in questo secondo caso è adottato un meccanismo per la misura carico e viene effettuata una selezione sulla base di tale misura. Per questa tipologia di algoritmi è importante evitare oscillazioni. Fanno parte di questa categoria gli algoritmi Join the Shortest Queue (che chiameremo Shortest Q), Join the Shortest Queue of Class (Shortest QofC) e gli algoritmi da noi creati Norma Infinito, Norma Quadratica e Norma Compatibilità.

Passiamo ora a descrivere, caso per caso, gli algoritmi sopra citati, partendo da quelli già presenti in letteratura, per finire con quelli da noi creati.

4.3.1 ALGORITMO RANDOM

L'algoritmo random come dice la parola stessa, indirizza le richieste in ingresso al Router in maniera casuale lunga le tre possibili uscite. Come si può immaginare questo algoritmo ha scarsa ottimalità in quanto non tiene conto in nessun modo, ne delle capacità dei singoli componenti, ne delle caratteristiche delle classi di Job. Il vantaggio però che si ottiene con un metodo random è quello di grande semplicità di implementazione e scarsa capacità computazionale richiesta.
All'interno del nostro progetto questo algoritmo è stato realizzato come segue:

```
scelta=rand()%3;
switch (scelta){
case(0):
outGateIndex =0;
break;
case(1):
outGateIndex = 1;
break;
case(2):
outGateIndex = 2;
break;
}
break;
```

Come vediamo attraverso una semplice istruzione switch case è possibile suddividere il carico delle richieste in ingresso in maniera random, con risultati però, come vedremo, non all'altezza.

4.3.2 ALGORITMO ROUND ROBIN

L'algoritmo Round Robin (RR) inoltra i Job in ingresso in maniera ripetitiva e ciclica sui vari outGateIndex. Essenzialmente quando un Job si presenta in Input al Router, viene inoltrato all' outGateIndex con valore ID superiore di un unità a quella utilizzata per il Job precedente. Questo vuol dire che se un Job viene inoltrato sull'uscita 0, ovvero alla macchina virtuale numero 1, il job successivo viene inoltrato alla virtual machine numero 2 e cosi via in maniera ciclica. In questo modo si sviluppa una forma, seppur molto grezza, di load balancing suddividendo le richieste sulle varie virtual machine. Come la tecnica precedente, anche l'algoritmo RR, non considera, ne le caratteristiche delle VM, ne i parametri delle classi di job. Il codice che realizza il meccanismo Round Robin è il seguente:

```
outGateIndex = rrCounter;
if(rrCounter==3)
{
rrCounter++;
outGateIndex=0;
}
rrCounter = (rrCounter + 1) % gateSize("out");
break;
```

rrCounter è la variabile che ci permette di creare il loop sui vari outGateIndex. Il metodo % gateSize("out") permette di ottenere un rrCounter di valore pari ad un numero associabile all'ID dell'outGateIndex. Quando rrCounter raggiunge il valore di gateSize+1 (nel nostro caso 3) si ricomincia dalla prima macchina virtuale. Anche in questo caso i risultati saranno scadenti anche se migliori rispetto alla politica Random.

4.3.3 ALGORITMO JOIN THE SHORTEST QUEUE (SHORTESTQ)

I primi due algoritmi analizzati sono considerati statici in quanto non considerano l'evoluzione nel tempo del modello. A questo proposito gli algoritmi che vedremo d'ora in avanti vanno ad interrogare il sistema riguardo le sue variabili di stato e a seconda del loro valore decidono la strada migliore a cui indirizzare la richiesta in ingresso.
Il primo algoritmo appartenente a questa seconda categoria che presentiamo è il Join The Shortest Queue. Come dice il nome stesso, questo algoritmo quando si presenta una richiesta in Input al router, la viene inoltra alla virtual machine con il minor numero di richieste attive. Tutto questo grazie a una serie di variabili che memorizzano, per ogni virtual machine il numero di Job (suddivisi per classe) attivi. Il metodo per incrementare e decrementare questi valori è semplice e descrivibile attraverso il seguente pseudo codice:

```
//Quando un Job viene inoltrato dal router
//verso una macchina virtuale i contatori corrispondenti
//vengono incrementati
 if(outGateIndex==0)
{

    JobVM1++;
    if(tipo==0)nrjobAVM1++;
    if(tipo==1)nrjobBVM1++;
    if(tipo==2)nrjobCVM1++;

}
 if(outGateIndex==1)
{
JobVM2++;

    if(tipo==0)nrjobAVM2++;
    if(tipo==1)nrjobBVM2++;
    if(tipo==2)nrjobCVM2++;

}
 if(outGateIndex==2)
{
JobVM3++;

    if(tipo==0)nrjobAVM3++;
    if(tipo==1)nrjobBVM3++;
    if(tipo==2)nrjobCVM3++;

}
```

e allo stesso modo quando arriva un messaggio di informazioni da parte del RouterInfo che ci notifica la terminazione di un determinato Job i contatori vengono decrementati nel seguente modo:

```
if(job->getSenderModuleId()==5)
{
JobVM1--;
outGateIndex=3;
if(tipo==0)nrjobAVM1--;
if(tipo==1)nrjobBVM1--;
if(tipo==2)nrjobCVM1--;
send(msg, "out", outGateIndex);
}
if(job->getSenderModuleId()==6)
{
JobVM2--;
outGateIndex=3;
if(tipo==0)nrjobAVM2--;
if(tipo==1)nrjobBVM2--;
if(tipo==2)nrjobCVM2--;
send(msg, "out", outGateIndex);
}
if(job->getSenderModuleId()==7) {
JobVM3--;
outGateIndex=3;
if(tipo==0)nrjobAVM3--;
if(tipo==1)nrjobBVM3--;
if(tipo==2)nrjobCVM3--;
send(msg, "out", outGateIndex);
}
```

Come si può vedere il metodo job->getSenderModuleId() restituisce l'ID riferito al RouterInfo che ha inoltrato il messaggio di informazioni, cosicché sia possibile decrementare i contatori della corrispondente macchina virtuale.
Il meccanismo di gestione delle variabili così descritto viene utilizzato da tutti gli algoritmi che considerano le caratteristiche prestazionali del sistema. Tornando alla descrizione dell'algoritmo Shortest Queue, esso ha un'implementazione molto semplice e pur non essendo un algoritmo ottimale, ha dimostrato di avere buone prestazioni rispetto ad altri algoritmi di complessità molto superiore [25]. Quando una richiesta si presenta in input al Router, si confrontano fra loro i valori delle variabili $JobVM_i$ e si inoltra tale richiesta alla virtual machine con valore minore di tale parametro. Così facendo il numero di richieste attive rimane sempre equamente distribuito, a livello numerico, tra le macchine virtuali presenti. Come si può facilmente intuire questo algoritmo considera esclusivamente la lunghezza della

coda di ogni macchina virtuale senza analizzare la classe dei Job presenti al loro interno. Per ovviare a questo problema abbiamo deciso di analizzare anche l'algoritmo Join The Shortest Queue of Class.

4.3.4 ALGORITMO JOIN THE SHORTEST QUEUE OF CLASS (SHORTESTQofC)

L'algoritmo Shortest Queue of Class è il primo algoritmo, tra quelli visti fino ad ora, che considera la caratteristica di Class-Based del nostro modello, ottenendo così buoni risultati. Il principio alla base di questo algoritmo è tuttavia molto semplice e si basa ancora una volta, sui valori dei contatori di job attivi sulle varie macchine virtuali. Supponiamo che in Input sia presente un Job di classe c, con $c \in C$ ($C = \{A, B, C\}$ insieme delle classi), il router controlla il contatore raffigurante il numero di Job_c attivi sulle varie macchine virtuali e indirizza la richiesta alla VM con valore di tale parametro minore. In caso di parità tra due o più macchine virtuali, il router considera, il numero di Job totali attivi sulle virtual machine, instradando la richiesta a quella con contatore a valore minore. In caso di ulteriore parità, si utilizza il metodo *random()* tra le candidate.
Per capire come questo meccanismo possa essere facilmente implementato, alleghiamo un piccolo spezzato di codice tratto dalla classe Router.cc:

```
//Supponimamo che che il Job in Input sia
//di classe A
if(nrjobAVM1<nrjobBVM2)
{
    if(nrjobAVM1<nrjobAVM3) outGateIndex=0;
    else if (nrjobAVM1>nrjobAVM3) outGateIndex=2;
    else //nrjobAVM1=nrjobAVM3
        {
        if(JobVM1>JobVM3)outGateIndex=2;
        else if(JobVM1<JobVM3)outGateIndex=0;
        else
        {
           routingClass=rand()%2;
           switch (routingClass)
           {
             case(0):
             outGateIndex =0;
```

```
            break;
            case(1):
            outGateIndex =2;
            break;
          }//fine switch random
        }//fine else JobVM1=JOBVM3
      }//fine else nrjobAVM1=nrjobAVM3
    }//fine if nrjobAVM1<nrjobAVM2
```

In questa porzione di codice viene analizzato solo il caso in cui il numero di Job_A della macchina virtuale numero 1 è minore del numero di $JobA$ della macchina virtuale numero 2. Ovviamente l'implemtentazione prosegue con tutti i restanti casi.
E' importante però capire come questo algoritmo, attraverso un meccanismo abbastanza semplice, riesca con ottimi risultati a dividere il carico in maniera uniforme tra le varie macchine. Il concetto alla base di tale operazione consiste nel fatto che così facendo i carichi delle varie macchine, risulteranno molto simili tra loro, pur senza considerare le caratteristiche proprie delle varie classi. Questo rende i tempi di attesa dei vari componenti minori e quindi permette di ottenere risultati che vedremo essere soddisfacenti.

I meccanismi di instradamento visti fino ad ora sono stati ampiamente implementati in altri progetti e descritti in letteratura. I prossimi algoritmi che andremo a proporre sono di nostra creazione e hanno come obiettivo finale, il miglioramento delle prestazioni dei loro predecessori. Questi nuovi meccanismi sono:

1. Norma Infinito
2. Norma Quadratica
3. Norma Compatibilità

Come si capisce semplicemente dall'analisi dei loro nomi, questi tre algoritmi, hanno una caratteristica comune, ovvero tutti e tre si basano sul concetto matematico di Norma. In algebra lineare, analisi funzionale e aree correlate della matematica, una norma è una funzione che assegna ad ogni vettore di uno spazio vettoriale, tranne lo zero, una lunghezza positiva. Matematicamente questo viene rappresentato come:

$$\begin{array}{rccc} \|\cdot\|: & X & \longrightarrow & [0,+\infty) \\ & x & \mapsto & \|x\| \end{array}$$

Figura 16: Definizione di Norma Matematica

che verifica le seguenti condizioni:

1. $\|x\| > 0 \, \forall x \in X$ e $\|x\| = 0$ se e solo se $x = 0$ (funzione definita positiva)
2. $\|\gamma x\| =\mid \gamma \mid \|x\|$ per ogni scalare γ (omogeneità)
3. $\|x + y\| \leq \|x\| + \|y\|$ per ogni $x, y \in X$ (disuguaglianza triangolare)

Nel nostro progetto questo significa che, se ad ogni macchina virtuale io assegno un vettore di carichi di lavoro (formato dai carichi dei singoli componenti della VM), la norma mi restituisce un valore riassuntivo di tale carico. Ogni algoritmo, attraverso una particolare funzione, converte il vettore dei carichi in un solo parametro

prestazionale che sarà utilizzato per instradare il Job in ingresso. A livello simulativo quindi, quando si presenta in Input una richiesta, il router calcola il valore della norma del carico che ogni macchina virtuale otterrebbe, se quel Job le fosse assegnato. La scelta di instradamento ricadrà sull'outGateIndex e quindi sulla virtual machine, con norma minore.
Prima di passare alla descrizione vera e propria dei vari algoritmi, introduciamo alcune definizioni che utilizzeremo nel proseguo del capitolo e all'interno delle implementazioni dei tre meccanismi sopra citati:

- $Carico_{k,m}$: identifica il carico di lavoro del componente k della macchina virtuale m e viene calcolato nel seguente modo

  $$Carico_{k,m} = \sum_{c \in C} NrJob_c * AvNumberRound_c * S_{c,k} con C = \{A, B, C\}$$

- $CaricoFuturo_{k,m}$: questo valore descrive una stima del carico di lavoro del componente k, se il Job (di classe c) in Input fosse instradato sulla macchina virtuale m. A livello implementativo

  $$CaricoFuturo_{k,m} = Carico_{k,m} + AvNumberRound_c * S_{c,k}$$

- $Carico_m$: identifica il carico di una determinata macchina virtuale m e viene calcolato come la somma dei carichi dei singoli componenti,

  $$Carico_m = Carico_{CPU,m} + Carico_{DISK,m} + Carico_{IO,m}$$

- $CaricoFuturo_m$: questo valore descrive una stima del carico di lavoro futuro della macchina m, se il Job in Input le fosse assegnato. Essp viene calcolato come somma dei carichi futuri dei singoli componenti,

  $$CaricoFuturo_m = CaricoFuturo_{CPU,m} + CaricoFuturo_{DISK,m} + CaricoFuturo_{IO,m} =$$

  $$= Carico_m + AvNumberRound_c * (S_{c,CPU} + S_{c,DISK} + S_{c,IO})$$

- $PercCarico_{k,m}$: identifica la percentuale di carico del componente k, rispetto al carico totale della macchina virtuale m,

  $$PercCarico_{k,m} = \frac{Carico_{k,m}}{Carico_m}$$

Tutti i valori di questi parametri, ad eccezione di $PercCarico_{k,m}$, sono espressi in secondi al fine di rappresentare il tempo medio che si impigherebbe a smaltire quel determinato carico di lavoro. Il calcolo delle variabili sopra elencate è essenziale per gli algoritmi che vedremo, in quanto in ognuno di essi verrà scelta come macchina virtuale a cui affidare il Job in ingresso, quella con $CaricoFuturo_m$ minore, al fine di garantire un buon load balancing nel sistema. La differenza tra i meccanismi che seguono sta in come essi matematicamente calcolano tale variabile.

4.3.5 ALGORITMO NORMA INFINITO

Questo particolare algoritmo, come detto, calcola la norma a partire dal vettore delle variabili $Carico_{k,m}$. La funzione che utilizza è tanto semplice quanto efficace. Per ogni macchina virtuale il router:

1. Calcola i valori delle variabili $Carico_{k,m}$
2. Calcola i valori delle variabili $CaricoFuturo_{k,m}$, come visto precedentemente
3. Calcola il $MaxCaricoFuturo_m$, dove $MaxCaricoFuturo_m = max_k \{CaricoFuturo_{k,m}\}$
4. Instrada il Job in ingresso verso la macchina virtuale m con minor carico futuro ovvero $min_m \{MaxCaricoFuturo_m\}$

A livello implementativo, quando due carichi si equivalgono, viene scelto in maniera casuale su quale macchina virtuale il Job verrà instradato. Per capire meglio il meccanismo di calcolo delle variabili $CaricoFuturo_{k,m}$ e $MaxCaricoFuturo_{k,m}$, utilizziamo una breve porzione di codice a titolo dimostrativo:

```
//Supponendo di avere in ingresso un JobA
//questo è il meccanismo per il calcolo del MaxCaricoVM1
MaxCaricoFuturoCPU_VM1=CaricoCPU_VM1+JOBA_CPU_NEED*nrGiriJobA;
if(MaxCaricoFuturoCPU_VM1>MaxCaricoVM1)
{

    MaxCaricoVM1=MaxCaricoFuturoCPU_VM1;

}
MaxCaricoFuturoDISK_VM1=CaricoDISK_VM1+JOBA_DISK_NEED*nrGiriJobA;
if(MaxCaricoFuturoDISK_VM1>MaxCaricoVM1)
{
```

```
    MaxCaricoVM1=MaxCaricoFuturoDISK_VM1;
}

MaxCaricoFuturoIO_VM1=CaricoIO_VM1+JOBA_IO_NEED*nrGiriJobA;
if(MaxCaricoFuturoIO_VM1>MaxCaricoVM1)
{
    MaxCaricoVM1=MaxCaricoFuturoIO_VM1;
}
//CaricoCPU_VM1+JOBA_CPU_NEED*nrGiriJobA rappresenta
//il
```
$CaricoFuturo_{k,m}$

Come vediamo $CaricoFuturo_{k,m}$ viene calcolato come somma tra $Carico_{k,m}$ (nel codice CaricoCPU_VM1) e l'incremento dovuto all'esecuzione del nuovo Job sulla macchina virtuale (nel codice JOBA_CPU_NEED*nrGiriJobA) come precedentemente descritto.

4.3.6 ALGORITMO NORMA QUADRATICA

L'algoritmo Norma Quadratica si basa su quella che viene comunemente definita Norma Due. Come il meccanismo precedente, la Norma Quadratica fa uso dei carichi di lavoro dei singoli componenti, per decidere il miglior instradamento possibile per la richiesta in Input al router. A differenza della Norma Infinito però, non considera solamente il $MaxCaricoFuturo_{k,m}$, ma tramite una particolare funzione riesce a considerare tutti i valori dei carichi di lavoro. Questa funzione utilizza il concetto di distanza euclidea che andiamo a presentare brevemente al fine di comprendere al meglio questo concetto.
In matematica, la distanza euclidea è la distanza fra due punti, ossia la misura del segmento avente per estremi i due punti. Usando questa formula come distanza, lo spazio euclideo diventa uno spazio metrico . La letteratura tradizionale si riferisce a questa metrica come metrica pitagorica. Per due punti in due dimensioni, $P = (p_x, p_y)$ e $Q = (q_x, q_y)$, la distanza euclidea in due dimensioni è calcolata come:

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Da questa formula è facile ricavare anche la distanza euclidea in tre dimensioni come la radice quadrata della somma dei quadrati delle tre componenti $(p_{dim} - q_{dim})$ con $dim \in \{x, y, z\}$.

Consideriamo ora di sostituire le differenze $(p_{dim} - q_{dim})$ con i valori dei carichi dei tre componenti di una certa macchina virtuale m allora la norma quadratica di tale VM, può essere calcolata nel seguente modo:

$$NormaQ = \sqrt{CaricoFut^2_{CPU,m} + CaricoFut^2_{DISK,m} + CaricoFut^2_{IO,m}}$$

dove $NormaQ = CaricoFuturo_m$ovvero rappresenta la variabile da analizzare per la scelta dell'outGateIndex.
Il risultato di questa formula considera equamente, tutti i carichi di lavoro dei singoli componenti. L'obiettivo finale di questo algoritmo è quindi dare ancora più importanza ai parametri caratterizzanti la classe del Job, durante la scelta del suo percorso di instradamento. Una volta calcolati i tre valori di $CaricoFuturo_m$ viene scelta per l'instradamento del Job in ingresso, ancora una volta, la macchina virtuale con il minor valore. Questo algoritmo, come si può capire, tiene in forte considerazione quella che viene considerata la $Lunghezza_c$ ovvero il tempo che impiegherebbe un Job di classe c ad essere elaborato in assenza di coda nei vari componenti, ovvero con un Arrival Time molto superiore al Service Time. La $Lunghezza_c$ è calcolata come:

$$Lunghezza_c = AvNumberRound_c * (S_{c,CPU} + S_{c,DISK} + S_{c,IO})$$

Il prossimo paragrafo descrive la norma compatibilità in cui abbiamo introdotto un indice per supportare il calcolo dei valori di carico delle singole macchine, cercando di migliorare i precedenti algoritmi.

4.3.7 ALGORITMO NORMA COMPATIBILITA'

L'ultimo algoritmo che vediamo della nostra carrellata è nato con l'intento di legare in maniera diretta, la scelta del persorso di instradamento migliore, con le caratteristiche proprie di ogni classe di Job. A differenza degli algoritmi precedenti, l'algortimo di Norma Compatibilità considera anche quanto l'arrivo di un Job, di una particolare classe, incide in termini di prestazionali, sui componenti della varie macchine virtuale. Tutto questo grazie ad un indice di nostra creazione chiamato $IndComp$ che rappresenta la compatibilità tra il Job in Input di classe c e il carico di lavoro già presente sulla macchina m. Matematicamente:

$$IndComp = \sum_{k=1}^{K} PercCarico_{k,m} * PercCaricoFuturo_{k,m}$$

dove $PercCaricoFuturo_{k,m}$viene calcolata alla stessa modo di $PercCarico_{k,m}$, ma considerando anche il Job in Input da instradare.
Il router si comporta nel seguente modo:

1. Calcola i valori delle variabili $PercCarico_{k,m} = \frac{Carico_{k,m}}{Carico_m}$ per ogni componente k, di ogni macchina m
2. Calcola i valori delle variabili $CaricoFuturo_{k,m}$
3. Calcola i valori delle variabili $CaricoFuturo_m$ come somma delle variabili (per ogni macchina m) calcolate al punto precedente
4. Calcola i valori delle variabili $PercCaricoFuturo_{k,m} = \frac{CaricoFuturo_{k,m}}{CaricoFuturo_m}$ per ogni componente k, di ogni macchina m
5. Calcola i valori della variabile $IndComp$ a seconda della classe c del Job in ingresso

Una volta ottenute queste informazioni è in grado, per ogni macchina virtuale, di calcolare il parametro $CaricoFuturo_m$ come segue:
$CaricoFuturo_m = Carico_m + AvNumberRound_c * (S_{c,CPU} + S_{c,DISK} + S_{c,IO}) * IndComp$

dove c è la classe del Job in Input. Come sempre il router sceglierà l'instradamento verso la virtual machine con $CaricoFuturo_m$ minore.
Così facendo se per esempio, la macchina virtuale numero uno avesse un'alta percentuale di carico della CPU e la classe del Job in ingresso (supponiamo classe A) dipende fortemente da quest'ultimo componente, la probabilità che la richiesta in Input venga dirottata su quella macchina virtuale risulterà bassa. Questo perchè l'indice di compatibilità tra la classe A e la VM1 sarà molto elevato, di conseguena anche il$CaricoFuturo_{VM1}$sarà grande in termini numerici e quindi con scarsa probabilità di essere il minore tra quelli in esame dal router.
Nei prossimi capitoli ci occuperemo di valutare e confrontare le prestazioni di tutti gli algoritmi descritti, analizzando alcuni casi particolari che ci faranno da guida per le conclusioni.

5 ANALISI PRESTAZIONALE DEGLI ALGORITMI

In questa sezione dell'elaborato di tesi andremo ad analizzare le prestazioni degli algoritmi visti fino ad ora, attraverso alcuni casi di studio che ci faranno da guida e che ci permetteranno, a fine capitolo, di trarre varie conclusioni sul comportamento degli algoritmi precedentemente descritti. Nella prima parte di questo capitolo però, abbiamo deciso di introdurre una guida alla creazione di una Run Configuration basata su file INI. Nella seconda parte, entreremo nel vivo dell'analisi prestazionale degli algoritmi, grazie ai dati statistici estrapolati dalle simulazioni effettuate sul nostro modello.

5.1 RUN CONFIGURATION

Per simulazione si intende una serie di run, a parametri variabili, atte a simulare la realtà attraverso il modello di riferimento. Per creare una simulazione occorre associare ad un determinato file INI, creato in precedenza, una run configuration. Quest'ultima viene creata attraverso la sequenza Run->RunConfiguration ottenendo così la seguente schermata:

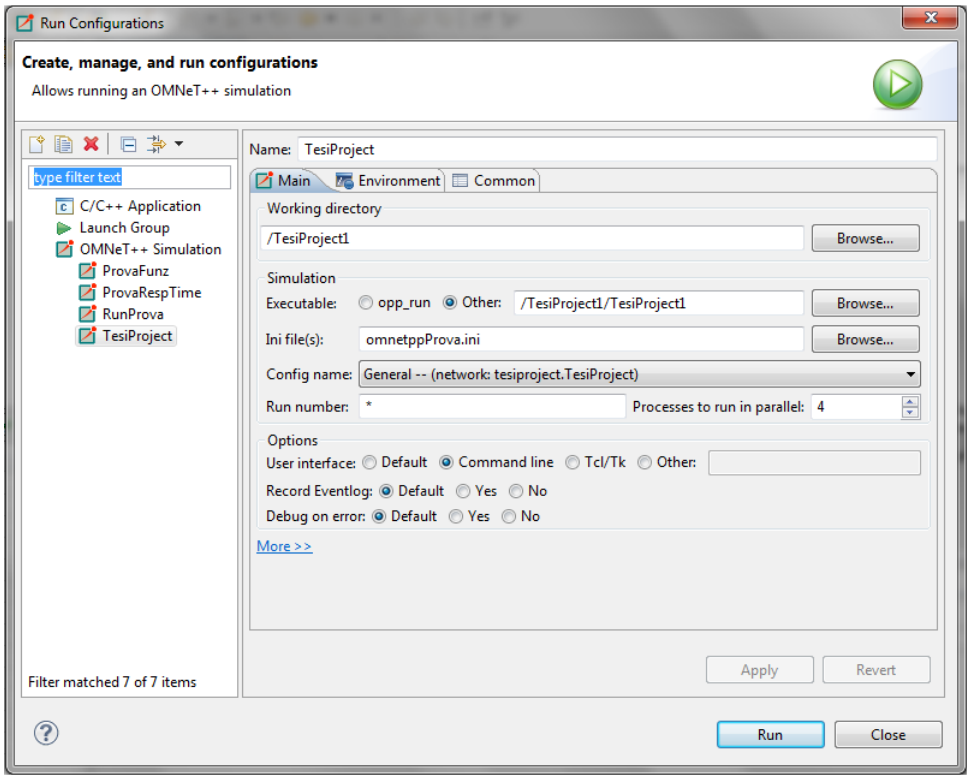

Figura 17: Esempio di Run Configuration

Nella parte sinistra troviamo il menù contenente le Run Configuration già create, mentre la parte centrale è l'espansione della configurazione selezionata e appena creata. Le voci che si sono rivelate fondamentali nel nostro progetto sono:

- Ini File(s): permette di associare un determinato File INI alla Run Configuration così da assorbirne le modifiche e le impostazioni
- Run number: permette di selezione quante run, tra quelle totali, si vogliono eseguire (con "*" vengono eseguite tutte le run)
- Process to run in parallel: a seconda della potenza della macchina su cui è installato Omnet++ è possibile eseguire più run in parallelo

- User Interface: permette di selezionare il metodo di iterazione con lo user. Default permette di vedere la simulazione come una vera e propria applicazione, dove i parametri vengono richiesti attraverso Text Box, mentre Command Line permette l'inserimento semplicemente da linea di comando
- Record Eventlog: permette la memorizzazione della variazione dei parametri studiati lungo il tempo in appositi file. Il valore Default indica che verrà adottata la politica dichiarata nel file INI

Dopo aver settato a piacimento e/o secondo necessità questi parametri si può avviare la simulazione e la nostra macchina dopo un lasso di tempo variabile a seconda della complessità, genererà una serie di valori, raggruppati in tabelle, che utilizzeremo per valutare le prestazioni degli algoritmi.

5.2 CASE STUDY

Dopo la breve descrizione dedicata alla creazione di una Run configuration, in questa seconda parte, descriveremo i casi di studio che abbiamo utilizzato per analizzare e studiare le prestazioni del sistema. Il primo Case Study è volutamente generale, al fine di fornire una visione completa di quelle che saranno le variabili principali del sistema. I restanti case study, seguiranno la procedura introdotta dal case study numero uno ma introdurranno delle modifiche ad una delle caratteristiche principali introdotte al Case Study 1, in modo da analizzare tutte le possibili sfaccettature delle richieste in ingresso. L'ultimo caso, infine, sarà un "riassunto" dei case study precedenti, atto a rappresentare una possibile situazione che potrebbe verificarsi nella realtà. In questo modo poi, analizzandone i risultati saremo in grado di trarre conclusioni riguardanti la bontà del nostro operato e l'effettivo grado di miglioramento degli algoritmi da noi introdotti.
Prima di tutto questo, abbiamo ritenuto opportuno, descrivere il test preliminare, a cui abbiamo sottoposto il nostro modello, al fine di verificarne il corretto funzionamento.

5.2.1 TEST DI FUNZIONAMENTO DEL SISTEMA

Prima di analizzare i casi di studio necessari alla valutazione delle prestazioni dei vari algoritmi, abbiamo ritenuto opportuno, come detto, inserire un paragrafo che descrivesse una parte, spesso sottovalutata, ma molto importante del nostro progetto, ovvero il testing del funzionamento del sistema. Quando si costruisce un ambiente di simulazione, prima di utilizzarlo per l'analisi prestazionale, è buona norma testarne il corretto funzionamento così da essere sicuri dei dati che verranno generati in output. Per fare ciò abbiamo deciso di utilizzare una tecnica molto usata che consiste nel creare un particolare caso di studio, in cui l'output è facilmente prevedibile, e osservare se effettivamente quello che otteniamo rispecchia le previsioni. Nel nostro progetto, abbiamo deciso di verificare che il caso in cui:

1. La coda di ogni componente sia nulla (per un tempo di simulazione sufficientemnte lungo): questo è possibile se l'Arrival Time è molto superiore al Service Time

2. La probabilità di round delle varie classi sia uguale a zero, ovvero che $AvNumberRound_c = 1 \; \forall c \in C$

il risultato che otteniamo è quello che l'Average Response Time di un determinato Job di classe c corrisponde esattamente alla somma delle singole $S_{c,k}$ ovvero che (nel caso del nostro progetto):

$$AvResponseTime_c = \sum_{k=1}^{3} S_{c,k} \; \forall c \in C$$

Abbiamo così deciso di testare questo specifico comportamento del simulatore partendo dai seguenti valori di $S_{c,k}$:

	CPU	$DISK$	IO
JOB_A	1	2	0.5
JOB_B	0.6	1	2
JOB_C	2	1.2	0.5

Risulta così facile intuire che i ResponseTime medi dovranno risultare pari rispettivamente a 3.5 secondi (Classe A), 3.6 secondi (Classe B) e 3.7 secondi (Classe C).
I risultati da noi ottenuti sono i seguenti (con un tempo di simulazione pari a 500000s):

- $AvResponseTime_A = 3.5008654s$
- $AvResponseTime_B = 3.60031124s$
- $AvResponseTime_C = 3.70008991s$

Possiamo quindi concludere che il nostro simulatore ha ottime probabilità di funzionare nella maniera corretta, in quanto i risultati ottenuti si avvicinano, in maniera considerevole, alle previsioni.

5.2.2 CASE STUDY 1

Il primo caso di studio che analizziamo ha, come anticipato nell'introduzione, caratteristiche molto generali. Abbiamo deciso di analizzare questo caso per primo, in quanto punto di partenza per migliorie future e fondamento per considerazioni generali sulle prestazioni dei vari algoritmi che analizzeremo. I casi successivi infatti, rappresenteranno comportamenti molto sbilanciati riguardo alcune caratteristiche del sistema che si possono otterenere apportando modifiche ai parametri ad esse riferiti.

Per prima cosa descriviamo attraverso i vari parametri che le caratterizzano, le classi di Job utilizzate per questa prima simulazione. Il primo parametro che andiamo a quantificare è la probabilità con cui un Job, creato dal componente Source, viene associato ad una delle tre classi. In questo primo caso di studio tale parametro $p_{Job,c}$ (con c che identifica la classe del Job) ha valore:

$$p_{Job,A} = p_{Job,cB} = p_{Job,C} = \frac{1}{3}$$

Ad ogni classe poi, abbiamo associato, secondo la matrice che segue, i valori $S_{c,k}$, rappresentanti il legame tra la Job Class e i componenti della macchina virtuale.

	CPU	$DISK$	IO
JOB_A	0.4	1	0.1
JOB_B	0.6	1.1	0.2
JOB_C	0.5	0.3	0.95

L'ultimo parametro settato è il numero medio di round per classe, ovvero quante elaborazioni, in media, occorrono ad un Job per terminare la sua esecuzione, secondo il seguente schema:

- $AvNumberRound_A = 2$
- $AvNumberRound_B = 2$
- $AvNumberRound_C = 2$

Grazie a questi ultimi valori, quindi, abbiamo calcolato la probabilità che un Job, di una determinata classe, effettui un Round per essere rielaborato, ottenendo i seguenti risultati $p_A = p_B = p_C = 0.5$.
L'ultimo parametro che rimane da analizzare è la scelta dei valori β da attribuire alla distribuzione dell'Arrival Time. Come descritto nei precedenti capitoli, questa decisione è fondamentale per capire il comportamento dei vari algoritmi [22] e proprio per questo abbiamo calcolato il valore limite, al di sotto del quale, la distribuzione non è più stabile ottenendo il seguente risultato:

$$\beta_{LIM} = 1.6$$

dove $\beta = \frac{1}{\eta}$, con η che identifica l'Arrival Rate.

Questo perchè la condizione affinchè la distribuzione sia stabile è:

$$max_k \left\{ \sum_{c=1}^{C} \eta_c D_{c,k} \right\} < 1$$

Per prima cosa, quindi, siamo andati a calcolare i vari $D_{c.k} = AvNumberRound_c * S_{c,k}$, ottenendo i seguenti risultati:

	CPU	$DISK$	IO
JOB_A	0.8	2	0.2
JOB_B	1.2	2.2	0.4
JOB_C	1	0.6	1.9

Ognuno di questi parametri è stato poi moltiplicato per $\eta_c = \eta p_{Job,c}$ (con $\beta = \frac{1}{\eta}$ che rappresenta l'incognita del nostro sistema) e poi, attraverso una suddivisione per componente, ne abbiamo effettuato la somma, ottenendo i seguenti valori:

- $D_{CPU} = 1\eta$
- $D_{DISK} = 1.6\eta$
- $D_{IO} = 0.83\eta$

Noi sappiamo che il massimo tra queste tre variabili deve essere minore di uno affinchè il sistema sia stabile cioè che:

$$D_{DISK} = 1.6\eta < 1$$

e questo è possibile per $\eta < 0.625$ ovvero per $\beta > 1.6$ da cui

$$\beta_{LIM} = 1.6$$

Questo primo vincolo può essere espresso anche dalla seguente tabella che mostra come, per valori di Arrival Time inferiori a 1.6 secondi, il Response Time medio, a prescindere dall'algoritmo selezionato, tende a crescere in maniera rapida (quasi esponenziale), mentre per valori superiori alla soglia, la diminuzione di tale parametro ha "velocità" molto minore.

$ArrivalTime$	$RandomAlg$
0.4	28393.32
0.7	12.02776
1	7.050618
1.6	5.276358
2	4.881608
4	4.255606

Come vediamo, analizzando l'algoritmo Random per esempio, con un $ArrivalTime = 0.4s$ (ovvero inferiore al livello di soglia) il valore di $AvResponseTime_B = 28393.32s$ ovvero la coda è in saturazione. Questa è una situazione da evitare assolutamente all'interno del sistema perchè è sintomo di livelli prestazionali insufficienti.
Per questo motivo abbiamo deciso di considerare, d'ora in avanti, solamente i campioni derivati da run aventi Arrival Time superiore o al massimo pari al valore soglia β_{LIM} (in questo primo caso di studio $\beta_{LIM} = 1.6s$).
Un altro parametro molto importante all'interno della nostra simulazione è $\alpha_{k,m}$, ovvero l'indice attraverso il quale è possibile descrivere la capacità prestazionale dei vari componenti delle tre macchine virtuali. Per semplicità questo parametro, nel caso di studio corrente e nei prossimi due, sarà posto uguale a uno in modo da non influire sulle prestazioni degli algoritmi. Vedremo poi, due particolari casi di studio, in cui a queste nove variabili, rappresentanti le singole $\alpha_{k,m}$, verranno dati dei valori differenti da uno e studieremo la conseguente evoluzione del sistema.
Una volta attuate queste precisazioni sui parametri del sistema, possiamo andare ad analizzare le tre tabelle che derivano dallo studio degli output del primo caso di studio. Ognuna di esse è costruita basandosi su una simulazione avente un numero di combinazioni pari a 35, ognuna delle quali è stata ripetuta quindici volte (numerosità del campione $n = 15$) al fine di garantire validità statistica alla prova. Il numero 35 è dato dal prodotto del numero di algoritmi analizzati (sette) per il numero di Arrival Timeβ (cinque) scelti. Ogni valore della tabella (espresso in secondi), rappresenta l'Average Response Time delle varie classi ottenuto attraverso la media delle quindici ripetizioni di ciascuna combinazione, con un tempo di simulazione pari a 500000 secondi.

	Random	RoundRobin	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
1,6	4,566610707	3,890363616	3,63857149	3,638512477	3,605198735	3,59074382	3,632216312
1,8	4,330109924	3,712609469	3,50203786	3,504909815	3,473386507	3,451022822	3,484697941
2	4,15379874	3,578584271	3,40058831	3,4014918	3,380409416	3,364672325	3,39013603
3	3,697679696	3,26366582	3,16096421	3,164709015	3,152587182	3,141172675	3,152889788
4	3,496316792	3,141569859	3,07606531	3,077605109	3,071276721	3,074033958	3,082233825

Tabella 1: AvResponseTime A Case Study 1

	Random	RoundRobin	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaCompl
1,6	5,274178067	4,648217212	4,389921331	4,352318987	4,313837687	4,318219559	4,295446584
1,8	5,052536262	4,479968805	4,258812385	4,225490877	4,188815984	4,19263831	4,176747622
2	4,15379874	4,366353694	4,166260737	4,137197392	4,103346853	4,121015022	4,092068666
3	4,455950754	4,064278511	3,951193941	3,937821897	3,91925447	3,923033936	3,912314373
4	4,264360501	3,937219738	3,873694653	3,86689739	3,861457553	3,858344471	3,852119106

Tabella 2: AvResponseTime B Case Study 1

	Random	RoundRobin	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaCompl
1,6	4,999651975	4,36998098	4,13512754	4,13779343	4,135291776	4,14103928	4,132528642
1,8	4,781481528	4,199038449	4,006968422	4,012469551	3,99854214	3,997764494	3,996693776
2	4,606244586	4,071127815	3,904143654	3,906219265	3,901543533	3,897585685	3,889700468
3	4,162712948	3,771636091	3,664049963	3,663221551	3,655007775	3,661583715	3,664240212
4	3,971230058	3,646565425	3,581377787	3,575024052	3,585263864	3,583282331	3,587530394

Tabella 3: AvResponseTime C Case Study 1

Come vediamo, i valori degli algoritmi Random e Round Robin risultano essere, in tutte e tre le tabelle (una per ogni classe di Job come detto), considerevolmente superiori rispetto ai rimanenti cinque algoritmi. Questo ci porta a pensare che, in termini prestazionali, non sono adatti ad un sistema di servizi Cloud che vuole fare delle prestazioni il suo punto di forza. Tutto ciò, è dovuto sicuramente al fatto che gli algoritmi Random e Round Robin sono statici, ovvero che non variano il loro comportamento al variare delle condizioni della rete, il che li porta da un lato ad avere una grande semplicità implementativa, ma dall'altro porta un peggiormanto in termini di prestazione.
Inoltre, per costruzione, essi non considerano nemmeno le caratteristiche delle varie classi, con la conseguente ulteriore perdita prestazionale. Per questo motivo l'analisi approfondita di questo caso di studio e dei successivi si baserà solamente sull'analisi dei seguenti algoritmi:

- Join The Shortest Queue
- Join The Shortest Queue of Class
- Norma Infinito
- Norma Quadratica
- Norma Compatibilità

A questo punto, è doveroso verificare che il numero n di ripetizioni da noi utilizzato sia stato sufficiente, a livello statistico, per garantire significatività dei risultati. Per questo motivo andiamo a calcolare gli intervalli di confidenza basati sull' $AvResponseTime$ degli algoritmi sopra citati. Nel proseguo dell'analisi di questo caso di studio, verranno analizzati le configurazioni aventi i seguenti Arrival Time:

- 1.8 secondi
- 2 secondi
- 3 secondi

Questa scelta è dovuta al fatto che con valori di Arrival Time elevati gli $AvResponseTime$ tendono al valore che si otterrebbe nel caso in cui la coda fosse nulla, a prescindere dall'algoritmo scelto. Un esempio di tutto ciò è dato dall'ultima riga ($ArrivalTime = 4s$) della Table 3 in cui, a prescindere dall'algoritmo selezionato, l'$AvResponseTime$ è circa $3.58s$.
Per il calcolo degli intervalli di confidenza utilizziamo una formula derivata dalla statistica e più precisamente dall'analisi delle distribuzioni normali (vedi sezione 2.2.2). Per prima cosa siamo andati a considerare la varianza come se fosse nota attribuendogli, grazie ad una sua proprietà, il seguente valore:

$\hat{\sigma}^2 = E[x^2] - E[x]^2$

per poi andare a calcolare i valori degli estremi dell'intervallo, avente probabilità $1 - \alpha = 0.95$, nel seguente modo:

$$(\overline{x} - z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}, \overline{x} + z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}})$$

ovvero

- $LowerEndPoint = \overline{x} - z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}$

- $UpperEndPoint = \overline{x} + z_{\frac{\alpha}{2}} \frac{\sigma}{\sqrt{n}}$

Considerando la varianza nota a priori e attribuendogli quel valore, abbiamo adottato una forte semplificazione che, in termini di precisione e affidabilità non è la migliore presente in letteratura ma garantisce comunque una buona base statistica di valutazione.
Tornando al caso in esame, abbiamo così calcolato i valori dei due estremi dell'intervallo per i valori di Arrival Time elencati in precedenza, suddivisi per algoritmo ottenendo le seguenti tabelle:

- $AvResponseTime_A$

	ShortestQueue		ShortestQueueOfClass	
	Lower	Upper	Lower	Upper
1,8	3,498178837	3,505896882	3,499852974	3,509966656
2	3,395396148	3,405780465	3,39585335	3,40713025
3	3,157836429	3,164091992	3,158847085	3,170570944

Tabella 4: Intervalli confidenza classe A Case Study 1

	NormaInfinito		NormaQuadratica		NormaComplementarietà	
	Lower	Upper	Lower	Upper	Lower	Upper
1,8	3,467166167	3,479606846	3,446101566	3,455944077	3,480940922	3,48845496
2	3,374507757	3,386311075	3,358799724	3,370544926	3,386579204	3,393692856
3	3,150006622	3,155325764	3,133993961	3,148351389	3,148503763	3,157275813

Tabella 5: Intervalli confidenza classe A Case Study 1

Come vediamo, i valori Lower degli intervalli degli algoritmi da noi creati, sono inferiori ai valori Upper degli algoritmi Shortest Queue e Shortest Queue of class. Questo significa che le campane delle distribuzioni normali di tali algoritmi non sono sovrapposte e per questo motivo possiamo dire che la numerosità del campione $n = 15$ garantistisce validità statistica alle considerazioni che produrremo sui valori dell'$AvResponseTime_c$. Nel caso opposto, ovvero se le campane fossero state sovrapposte in maniera considerevole, avrebbe significato che i valori di $AvResponseTime_c$ non rispecchiavano la realtà e che quindi avremmo dovuto aumentare n per poter trarre delle conclusioni veritiere, soprattutto in termini statistici.

- $AvResponseTime_B$

	ShortestQueue		ShortestQueueOfClass	
	Lower	Upper	Lower	Upper
1,8	4,252318746	4,265306025	4,217088772	4,233892982
2	4,16069097	4,171830505	4,13077479	4,143619995
3	3,945321689	3,957066193	3,932494956	3,943148838

Tabella 6: Intervalli confidenza classe B Case Study 1

	NormaInfinito		NormaQuadratica		NormaComplementarietà	
	Lower	Upper	Lower	Upper	Lower	Upper
1,8	4,184795414	4,192836553	4,186176509	4,19910011	4,17004327	4,183451974
2	4,096695998	4,109997709	4,114989478	4,127040565	4,085187411	4,098949921
3	3,913187837	3,925943431	3,917648207	3,928419666	3,905763692	3,918865053

Tabella 7: Intervalli confidenza classe B Case Study 1

Anche in questo caso, come nel precedente,i valori Lower degli intervalli degli algoritmi da noi creati, sono inferiori ai valori Upper degli algoritmi Shortest Queue e Shortest Queue of class. Per questo motivo possiamo trarre le stesse conclusioni del punto precedente e dire che il valore di n, è sufficientemente grande da garantire significatività statistica dei risultati.

- $AvResponseTime_C$

	ShortestQueue		ShortestQueueOfClass	
	Lower	Upper	Lower	Upper
1,8	3,998003711	4,015933133	3,994484901	4,030454202
2	3,898539505	3,909747803	3,893991498	3,918447033
3	3,658416254	3,669683672	3,656910203	3,669532898

Tabella 8: Intervalli confidenza classe C Case Study 1

	NormaInfinito		NormaQuadratica		NormaComplementarietà	
	Lower	Upper	Lower	Upper	Lower	Upper
1,8	3,990243644	4,006840637	3,989381981	4,006147007	3,990251337	4,003136215
2	3,896373915	3,906713151	3,893922941	3,901248429	3,884261003	3,895139932
3	3,650542286	3,660429516	3,658770739	3,664396691	3,659718864	3,668761561

Tabella 9: Intervalli confidenza classe C Case Study 1

> Anche nell'ultimo caso, ovvero con Job di classe C, i valori Lower degli intervalli degli algoritmi da noi creati, sono inferiori ai valori Upper degli algoritmi Shortest Queue e Shortest Queue of class. n quindi è sufficientemente elevata per garantire la significatività dei dati in output.

Dopo aver verificato la correttezza del parametro n, rappresentante il numero di run per ogni configurazione, possiamo trarre alcune considerazioni sui valori di $AvResponseTime$ riguardanti i cinque algoritmi analizzati.
Studiando gli elementi contenuti nelle tre tabelle, rappresentanti gli $AvResponseTime$ divisi per classe, ci accorgiamo come non ci sono, a primo impatto, algoritmi nettamente dominanti sugli altri, anche se gli algoritmi basati su Norma risultano più efficienti rispetto agli altri due. Per verificare che questa nostra ultima considerazione è fondata, abbiamo deciso di fondere le tre tabelle in una visione complessiva del sistema. Per fare ciò abbiamo utilizzato la variabile $p_{job,c}$ (ovvero la probabilità che un job appartenga alla classe c), grazie alla quale siamo andati a calcolare la media pesata di ciascun valore di Response Time ($AvResponseTimeWeight$) nel seguente modo:
$X_{ArrTime,alg} = X_{ArrTime,alg,A} * p_{job,A} + X_{ArrTime,alg,B} * p_{job,B} + X_{ArrTime,alg,C} * p_{job,C}$
dove $X_{ArrTime,Alg,c}$rappresenta l'$AvResponseTime_c$ ottenuto con l'algoritmo identificato dalla variabile Alg e con un arrival time pari a $ArrTime$. La tabella risultante è la seguente(Table 10):

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaCompl
1,6	4,054540119	4,042874964	4,0181094	4,016667553	4,020063846
1,8	3,922606222	3,914290081	3,886914877	3,880475209	3,886046446
2	3,823664233	3,814969486	3,795099934	3,794424344	3,790635055
3	3,592069372	3,588584154	3,575616476	3,575263442	3,576481458
4	3,510379251	3,50650885	3,50599938	3,505220253	3,507294441

Tabella 10: Average ResponseTime Pesato Case Study 1

Da un analisi dettagliata dei valori della Table 10, risulta chiaro ciò che avevamo precedentemente concluso, ovvero che gli algoritmi da noi creati, in media, hanno risultati migliori in termini di prestazioni. Questo è dovuto al fatto che gli algoritmi Shortest Queue e Shortest Queue of class, non considerano le caratteristiche delle classi dei vari Job, ma solamente la lunghezza della coda di una determinata macchina virtuale da un lato e il numero di Job per classe nelle varie code dall'altro. Se effettuassimo una simulazione con un numero di ripetizioni ancora maggiore (circa cinquanta) questa caratteristica sarebbe ancor più evidente in quanto miglioraremmo la significatività statistica dei nostri dati. All'interno degli algoritmi basati su norma invece, è difficile eleggere un fantomatico vincitore, in quanto i valori di Response Time di riferimento sono molto simili tra loro e non permettono di differenziare i risultati.

5.2.3 CASE STUDY 2

Il secondo case study che andiamo ad analizzare, rappresenta il caso in cui le probabilità che un determinato job venga assegnato ad una classe rispetto ad una altra sono differenti. In questo secondo caso abbiamo utilizzato:

- $p_{job,A} = 0.7$
- $p_{job,B} = 0.15$
- $p_{job,C} = 0.15$

Questo in termini di implementazione è possibile utilizzando un generatore di numeri casuali che, a seconda del valore emesso, fa ricadere la variabile decisionale all'interno di un ramo piuttosto che di un altro, assegnando il Job alla classe corrispondente. Per una migliore comprensione, di seguito, descriviamo attraverso pseudo codice questo meccanismo:

```
int NumCasuale = rand() %10;
if(NumCasuale<7){

    job->setKind(0);
    job->setPriority(0);

}
else {


    int NumCasuale_2 = rand() %2;
    switch (NumCasuale_2)
    {
    case(0):
        job->setKind(1);
        job->setPriority(0);
    break;
    case(1):
        job->setKind(2);
        job->setPriority(0);
    break;
    }

}
```

Se la variabile NumCasuale verrà generata con un valore minore di sette ($p_{job,A} = 0.7$) allora il Job verrà assegnato alla classe A, altrimenti con probabilità del quindici percento alla classe B o C.
Con questo study case vogliamo simulare il caso in cui le richieste in Input ad un Cloud Provider abbiano grandi possibilità di essere dello stesso tipo, per vedere come si comporta il nostro sistema. Le rimanenti caratteristiche sono rimaste invariate rispetto al primo caso, ma risulta comunque necessario ricalcolare il β_{LIM}. La tabella di partenza, raffigurante i vari $D_{c.k} = AvNumberRound_c * S_{c,k}$, rimane la stessa:

	CPU	$DISK$	IO
JOB_A	0.8	2	0.2
JOB_B	1.2	2.2	0.4
JOB_C	1	0.6	1.9

alla quale andiamo a moltiplicare, per ogni valore al suo interno, il coefficiente $\eta_c = \eta p_{Job,c}$ (con $\beta = \frac{1}{\eta}$ che rappresenta l'incognita del nostro sistema). Attraverso una suddivisione per componente, poi, ne abbiamo effettuato la somma, ottenendo i seguenti valori:

	ShortestQueue		ShortestQueueOfClass	
	Lower	Upper	Lower	Upper
2	3,352878595	3,358252087	3,339973616	3,35555872
2,5	3,21174241	3,22750917	3,206950664	3,21972424
3	3,135423988	3,143117686	3,135439165	3,14218345

Tabella 11: Intervalli confidenza classe A Case Study 2

- $D_{CPU} = 0.7 * 0.8\eta + 0.15 * 1.2\eta + 0.15 * 1\eta = 0.89\eta$
- $D_{DISK} = 0.7 * 2\eta + 0.15 * 2.2\eta + 0.15 * 0.6\eta = 1.82\eta$
- $D_{IO} = 0.7 * 0.2\eta + 0.15 * 0.4\eta + 0.15 * 1.9\eta = 0.485\eta$

Noi sappiamo che il massimo tra queste tre variabili deve essere minore di uno affinchè il sistema sia stabile cioè che:

$$D_{DISK} = 1.82\eta < 1$$

e questo è possibile per $\eta < 0.549$ ovvero per $\beta > 1.82$ da cui

$$\beta_{LIM} = 1.82$$

Come il caso precedente anche in questo secondo, analizzeremo il comportamento del sistema per valori maggiori o uguali a $\beta_{LIM} = 1.82$ concentrando la nostra attenzione sui seguenti algoritmi:

- Join The Shortest Queue
- Join The Shortest Queue of Class
- Norma Infinito
- Norma Quadratica
- Norma Compatibilità

Anche in questo caso, il primo passaggio, è garantire veridicità statistica ai risultati e per fare ciò siamo andati a verificare che la numerosità del campione sia sufficientemente elevata. Riportiamo di seguito i valori degli estremi dei vari intervalli di confidenza:

	NormaInfinito		NormaQuadratica		NormaComplementarietà	
	Lower	Upper	Lower	Upper	Lower	Upper
2	3,33217743	3,337726088	3,330271605	3,336071496	3,331935727	3,341523606
2,5	3,20158171	3,207492814	3,198453674	3,205315824	3,204028819	3,207729326
3	3,12703828	3,136335884	3,121734222	3,128657821	3,128568227	3,13577775

Tabella 12: Intervalli confidenza classe A Case Study 2

	ShortestQueue		ShortestQueueOfClass	
	Lower	Upper	Lower	Upper
2	4,122085217	4,138472309	4,138114795	4,155699244
2,5	4,004438309	4,026387298	4,015317377	4,030460329
3	3,930518153	3,948968236	3,93899154	3,95830237

Tabella 13: Intervalli confidenza classe B Case Study 2

	NormaInfinito		NormaQuadratica		NormaComplementarietà	
	Lower	Upper	Lower	Upper	Lower	Upper
2	4,045204306	4,064411586	4,071548734	4,089914427	4,047958317	4,065146293
2,5	3,942834935	3,959410087	3,974056925	3,984280149	3,94715845	3,966166656
3	3,886007791	3,911306761	3,903986691	3,92154022	3,877555127	3,896647057

Tabella 14: Intervalli confidenza classe B Case Study 2

	ShortestQueue		ShortestQueueOfClass	
	Lower	Upper	Lower	Upper
2	3,846043039	3,869225845	3,844642115	3,863604255
2,5	3,715626351	3,748690017	3,717317611	3,738435718
3	3,639693049	3,654911804	3,639072407	3,649940433

Tabella 15: Intervalli confidenza classe C Case Study 2

	NormaInfinito		NormaQuadratica		NormaComplementarietà	
	Lower	Upper	Lower	Upper	Lower	Upper
2	3,832219989	3,844001083	3,83211734	3,846459964	3,832824175	3,845346501
2,5	3,705178232	3,718989078	3,701460765	3,718908599	3,694928828	3,712342176
3	3,623902136	3,637085897	3,622454855	3,638736571	3,625701606	3,637621789

Tabella 16: Intervalli confidenza classe C Case Study 2

Come vediamo, questo secondo caso di studio ha risultati simili al primo, infatti anche in questa occasione i nostri tre algoritmi hanno campane, raffiguranti la distribuzione normale, che non sovrappongono tra loro.
Garantita l'accuratezza dei dati, passiamo ora all'analisi delle tabelle (di seguito elencate) degli $AvResponseTime$, divisi per classe, di questo particolare caso di studio

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
1,82	3,432580825	3,426096645	3,417353269	3,415395261	3,423236892
2	3,355565341	3,349766167	3,334951759	3,333171551	3,336729666
2,5	3,21962579	3,213337451	3,204537263	3,201884749	3,205879072
3	3,139270837	3,138811306	3,131687081	3,125196021	3,132172989
4	3,070518518	3,066055977	3,065156728	3,061505931	3,06614739

Tabella 17: AvResponseTime A Case Study 2

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
1,82	4,204211827	4,222103393	4,12782985	4,151229164	4,1233521
2	4,130278763	4,14690702	4,054807946	4,080731581	4,056552305
2,5	4,015412804	4,022888853	3,951122511	3,979168537	3,956662553
3	3,939743195	3,948646955	3,898657276	3,912763455	3,887101092
4	3,877505555	3,879479467	3,850610195	3,850463086	3,85256477

Tabella 18: AvResponseTime B Case Study 2

	ShortestQ	ShortestQC	NormaInf	NormaQ	NormaC
1,82	3,898838459	3,91864167	3,933285368	3,9261702	3,922493813
2	3,857634442	3,854123185	3,838110536	3,839288652	3,839085338
2,5	3,732158184	3,727876665	3,712083655	3,710184682	3,703635502
3	3,647302426	3,64450642	3,630494016	3,630595713	3,631661697
4	3,562159854	3,559563988	3,566506573	3,575607599	3,563345026

Tabella 19: AvResponseTime C Case Study 2

Da un'attenta lettura di queste tabelle ci rendiamo subito conto che gli algoritmi da noi creati risultano avere dei tempi di risposta medi leggermente inferiori ai restanti due algoritmi. Questo conferma che i "nostri" meccanismi, in termini di risultati, stanno restituendo le risposte che speravamo. Per avvalorare ulteriormente la nostra tesi, abbiamo costruito la tabella identificativa dell'$AvResponseTimeWeight$, in cui ogni elemento $X_{ArrTime,alg}$ è dato dalla seguente equazione: $X_{ArrTime,alg} = X_{ArrTime,alg,A} * p_{job,A} + X_{ArrTime,alg,B} * p_{job,B} + X_{ArrTime,alg,C} * p_{job,C}$
dove A, B, C rappresentano le nostre tre classi di Job.
La tabella (Table 20) che abbiamo ottenuto è la seguente:

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaCompl
1,82	3,615164121	3,609846078	3,600014571	3,602386587	3,603142711
2	3,539244385	3,532190848	3,52129067	3,52202312	3,52522308
2,5	3,404607034	3,40081771	3,394553676	3,395388974	3,395413392
3	3,331346429	3,331807587	3,323153651	3,31934109	3,32173551
4	3,265312774	3,260229036	3,258177225	3,256964755	3,258689642

Tabella 20: AvResponseTimePesato Case Study 2

Come si può vedere, anche in questo caso i risultati confermano quanto detto precedentemente, infatti un Job in ingresso, guidato dagli algoritmi Norma Infinito, Norma Quadratica e Norma Compatibilità impiegherà un lasso di tempo inferiore ad essere elaborato, rispetto agli altri due algoritmi. Questo perchè, proprio come nel caso precedente, questi algoritmi agiscono con un approccio che considera, seppur in modi diversi tra loro, le caratteristiche delle classi di Job riuscendo ad ottenere così, un piccolo miglioramento nelle prestazioni.

5.2.4 CASE STUDY 3

In questo caso di studio, la caratteristica del sistema che andremo a modificare è la "lunghezza" delle classi di Job, senza intaccare i restanti parametri. Così facendo, aumentando particolarmente il valore di una classe, otterremo tutta una serie di richieste, la cui durata, è molto maggiore rispetto alle altre. Per fare ciò abbiamo modificato i valori di $S_{c,k}$ nel seguente modo:

	CPU	$DISK$	IO
JOB_A	4	0.4	0.2
JOB_B	0.6	1.1	0.2
JOB_C	0.5	0.3	0.95

Così facendo, i Job di classe A avranno lunghezza totale pari a $4.6s$ e richiederanno un maggior tempo di elaborazione rispetto alle richieste appartenenti alle restanti class. Ovviamente cambiando in maniera così radicale le caratteristiche identificative della classe, varia anche il valore limite di stabilità del sistema ovvero il β_{LIM}. Per questo motivo lo andiamo a ricalcolare. Più precisamente partendo dalla seguente tabella, raffigurante i vari $D_{c.k} = AvNumberRound_c * S_{c,k}$:

	CPU	$DISK$	IO
JOB_A	8	0.8	0.4
JOB_B	1.2	2.2	0.4
JOB_C	1	0.6	1.9

andiamo a moltiplicare, per ogni valore al suo interno, il coefficiente $\eta_c = \eta p_{Job,c}$ (con $\beta = \frac{1}{\eta}$ che rappresenta l'incognita del nostro sistema). Attraverso una suddivisione per componente, poi, ne abbiamo effettuato la somma, ottenendo i seguenti valori:

- $D_{CPU} = 0.33 * 8\eta + 0.33 * 1.2\eta + 0.33 * 1\eta = 3.4\eta$
- $D_{DISK} = 0.33 * 0.8\eta + 0.33 * 2.2\eta + 0.33 * 0.6\eta = 1.2\eta$
- $D_{IO} = 0.33 * 0.4\eta + 0.33 * 0.4\eta + 0.33 * 1.9\eta = 0.9\eta$

Noi sappiamo che il massimo tra queste tre variabili deve essere minore di uno affinchè il sistema sia stabile cioè che:

$$D_{DISK} = 3.4\eta < 1$$

e questo è possibile per $\eta < 0.294$ ovvero per $\beta > 3.4$ da cui

$$\beta_{LIM} = 3.4$$

Per questo motivo, per questo esperimento useremo i seguenti valori:

- 3.4 secondi
- 3.7 secondi
- 4 secondi
- 4.5 secondi
- 5 secondi

Così facendo abbiamo ottenuto le tre seguenti tabelle, ognuna raffigurante i valori di $AvResponseTime_c$ del terzo Case Study:

	ShortestQ	ShortestQC	NormaInf	NormaQuad	NormaComp
3,4	10,2516814	9,868991623	10,33058287	10,31590407	9,871846475
3,7	10,05959984	9,742736693	10,13563115	10,0815715	9,728290997
4	9,311774695	9,657230512	9,963237814	9,928898277	9,619335543
4,5	9,735057052	9,52633661	9,732799329	9,722650643	9,501261794
5	9,639835905	9,428123361	9,609526815	9,623040061	9,417274469

Tabella 21: AvResponseTime A Case Study 3

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
3,4	4,717270285	4,868052332	4,849856795	4,832676496	4,602129952
3,7	4,539963717	4,692769625	4,651567758	4,638998305	4,44560026
4	4,419489691	4,541153174	4,484598505	4,483100693	4,33321023
4,5	4,269769646	4,36086709	4,292648969	4,321092611	4,191367567
5	4,161150913	4,236123555	4,165689506	4,186587709	4,097960241

Tabella 22: AvResponseTime B Case Study 3

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
3,4	4,4202126	4,602214183	4,770026012	4,641639962	4,407036238
3,7	4,25280646	4,4155351	4,521924559	4,427044954	4,236543355
4	4,135218388	4,257687794	4,336240941	4,266966085	4,114064453
4,5	3,980392446	4,074059431	4,106200483	4,067168328	3,960991638
5	3,866530743	3,948577087	3,972045846	3,928841923	3,852291177

Tabella 23: AvResponseTime C Case Study 3

Come si può vedere dalle seguenti tabelle ci sono due concetti molto interessanti da sviluppare:

1. In questo caso di studio così particolare, siamo in grado di eleggere un "vincitore" ovvero siamo in grado di dire, per la prima volta, che esiste un algoritmo migliore rispetto agli altri, ovvero l'algoritmo Norma Compatibilità. Questo perchè, grazie al coefficiente che lo contraddistingue, è in grado di indirizzare il job in Input verso la VM, avente le richieste in coda più compatibili con quella in ingresso. Negli altri casi analizzati fino ad ora, questa caratteristica non era molto evidente, in quanto gli indici di compatibilità risultavano numericamente poco determinanti al fine del calcolo del percorso migliore.

2. I restanti algoritmi da noi creati non si comportano come speravamo, sopratutto la Norma Infinito, in quanto i valori di Average Response Time risultano molto maggiori rispetto al "vincitore" in tutte e tre le tabelle. Abbiamo così scoperto il loro punto debole, ovvero essi "soffrono" in particolar modo il caso in cui, una classe di richieste si rivela di lunghezza molto maggiore rispetto alle altre e fa crescere il tempo di coda dei job.

Queste considerazioni, che eleggono l'algoritmo Norma Compatibilità come "vincitore", sono garantite a livello statistico dal fatto che, con una numerosità del campione pari a quindici ($n = 15$), la campana generata dalla distribuzione normale che lo caratterizza, non sovrappone con nessuna delle altre quattro (testato come nei casi di studio precedenti). Questo vuol dire che, con una probabilità

del 95% e a parità di restanti condizioni, l' $AvResponseTime$ generato dall'algoritmo Norma Compatibilità è minore rispetto ad uno qualsiasi generato dai restanti algoritmi.

5.2.5 CASE STUDY 4

In questo caso di studio, a differenza dei precedenti, abbiamo modificato i valori del parametro raffigurante i singoli componenti delle macchine virtuali ovvero l' $\alpha_{k,m}$. Questo indicatore ci caratterizza, in termini prestazionali i tre moduli presenti su ogni VM: la CPU, il Disk e l'IO. Tanto più questa caratteristica avrà un valore basso tanto più sarà minore il valore di τ, ovvero il parametro della distribuzione del $ServiceTime$ di quel componente. In altre parole, riducendo $\alpha_{k,m}$ si riduce il tempo che un determinato componente k impiegherà ad elaborare la richiesta in ingresso. Nei casi visti fino ad ora, il parametro alfa era caratterizzato dalla seguente formula

$\alpha_{k,m} = 1 \; \forall k \in K, \forall m \in M$ con $K = \{CPU, Disk, IO\}$ e $M = \{VM1, VM2, VM3\}$
mentre nel caso di studio in esame:

	CPU	$DISK$	IO
$VM1$	0.5	0.8	0.9
$VM2$	0.9	0.6	0.7
$VM3$	0.7	0.9	0.4

Questo significa ad esempio, che la CPU della VM1 impiegherà la metà del tempo ad elaborare una richiesta in ingresso, rispetto ai casi precedenti. Per questo motivo gli $AvResponseTime$ che otterremo saranno molto inferiori, a parità di restanti condizioni rispetto ai Case Study visti fino ad ora.
A livello implementativo, affinchè tutto questo sia possibile, abbiamo dovuto apportare delle modifiche agli algoritmi da noi creati in modo tale da garantire che il parametro $\alpha_{k,m}$fosse adeguatamente considerato. Questo vuol dire che, ogni qual volta, il sistema debba calcolare la variabile $Carico_{k,m}$(o$CaricoFuturo_{k,m}$,vedi paragrafo 4.3), ovvero quanto tempo (calcolato in secondi) impiega un determinato componente k, ad elaborare una richiesta in ingresso, abbiamo considerato anche alfa ottenendo la seguente formula matematica:

$$Carico_{k,m} = \sum_{c} \alpha_{k,m} * AvNumberRound_c * S_{c,k}$$

dove c rappresenta la classe.
I risultati che abbiamo ottenuto (con una numerosità del campione $n = 30$) e che analizzeremo in seguito, sono garantiti, a livello statistico, dal test sugli Intervalli di Confidenza che già abbiamo visto all'opera precedentemente. Per non appesantire troppo la lettura del nostro trattato non alleghiamo i risultati di tale test, ma passiamo direttamente all'analisi delle tabelle raffiguranti gli $AvResponseTime$ divisi per classe:

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
1,6	2,362651521	2,363559189	2,139440098	2,163593739	2,349104793
1,8	2,316683782	2,312857464	2,08651794	2,107673461	2,311862006
2	2,279416223	2,277471331	2,04327053	2,060276404	2,279959419
3	2,195743488	2,195232762	1,939898274	1,947940299	2,19577795

Tabella 24: AvResponseTime A Case Study 4

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
1,6	3,02121935	2,994919349	2,90153777	2,879529106	2,967502701
1,8	2,969092123	2,949169488	2,861120701	2,839156202	2,925512307
2	2,932169674	2,916243949	2,828337254	2,808248618	2,896656504
3	2,84877467	2,842029447	2,748075337	2,7337796	2,832796393

Tabella 25: AvResponseTime B Case Study 4

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
1,6	2,658588989	2,663688741	2,411810234	2,406159116	2,642441849
1,8	2,603471521	2,60851937	2,352433606	2,343439292	2,592782232
2	2,566433703	2,571542177	2,303944072	2,293540131	2,559912942
3	2,480988024	2,482790302	2,179051178	2,169860641	2,476423075

Tabella 26: AvResponseTime C Case Study 4

In questo caso notiamo come gli algoritmi Norma Infinito e Norma Quadratica ottengano dei risultati nettamente superiori rispetto agli

altri meccanismi. Questo è possibile grazie al fatto che in questi due algoritmi viene considerato il livello prestazionale dei singoli componenti delle tre macchine virtuali, cosa che non accade per gli algoritmi Shortest Queue e Shortest Queue of Class. Analizzando poi il meccanismo Norma Compatibilità, probabilmente il coefficiente alla base del suo funzionamento, viene "numericamente" surclassato di valore dal peso dei coefficienti $\alpha_{k,m}$, non riuscendo così ad ottenere, i risultati da noi sperati.

5.2.6 CASE STUDY 5

In questo ultimo caso di studio, abbiamo cercato di riassumere tutti i Case Study visti in precedenza, modificando così tutti quei parametri che abbiamo visto fino ad ora. Per questo motivo abbiamo introdotto le seguenti modifiche al sistema:

1. $\alpha_{k,m}$: i valori che rappresentano le capacità prestazionali dei componenti in questo case study hanno valore pari a

	CPU	$DISK$	IO
$VM1$	0.6	0.8	0.9
$VM2$	0.9	0.6	0.7
$VM3$	0.7	0.9	0.6

2. $p_{job,c}$: le probabilità che una richiesta in ingresso venga associata ad una determinata classe saranno differenti, in particolare $p_{job,A} = 0.2$, $p_{job,B} = 0.5$ e $p_{job,C} = 0.3$

3. $AvNumberRound_c$: la classe A sarà caratterizzata da Job aventi $AvNumberRound_A = 2$, B da $AvNumberRound_B = 1$ e C da $AvNumberRound_C = 2$

4. $S_{c,k}$: per garantire "lunghezze" diverse alle tre classi di richieste andiamo a definire la seguente tabella raffigurante i $S_{c,k}$:

	CPU	$DISK$	IO
JOB_A	1	0.8	0.5
JOB_B	0.6	1.1	0.2
JOB_C	0.2	0.3	0.5

Ovviamente andando a modificare i valori di queste ultime tre caratteristiche, varierà anche la misura del limite di stabilità del sistema

ovvero il β_{LIM}. Per questo motivo, andiamo ad aggiornarlo partendo dalla tabella raffigurante i $D_{c.k}$ con $D_{c.k} = AvNumberRound_c * S_{c,k}$:

	CPU	$DISK$	IO
JOB_A	2	1.6	0.5
JOB_B	0.6	1.1	0.2
JOB_C	0.4	0.6	1

alla quale andiamo a moltiplicare, per ogni valore al suo interno, il coefficiente $\eta_c = \eta p_{Job,c}$ (con $\beta = \frac{1}{\eta}$ che rappresenta l'incognita del nostro sistema). Attraverso una suddivisione per componente, poi, ne abbiamo effettuato la somma, ottenendo i seguenti valori:
$D_{CPU} = 0.2 * 2\eta + 0.5 * 0.6\eta + 0.3 * 0.4\eta = 0.82\eta$
$D_{DISK} = 0.2 * 1.6\eta + 0.5 * 1.1\eta + 0.3 * 0.6\eta = 1.05\eta$
$D_{IO} = 0.2 * 0.5\eta + 0.5 * 0.2\eta + 0.3 * 1\eta = 0.5\eta$
Noi sappiamo che il massimo tra queste tre variabili deve essere minore di uno, affinchè il sistema sia stabile, cioè che:

$$D_{DISK} = 1.05\eta < 1$$

e questo è possibile per $\eta < 0.95$ ovvero per $\beta > 1.05$ da cui

$$\beta_{LIM} = 1.05$$

Per questo motivo, per questo esperimento useremo i seguenti valori:

- 1.05 secondi
- 1.2 secondi
- 1.4 secondi
- 1.6 secondi
- 1.8 secondi

Abbiamo deciso di porre la numerosità del campione pari a 30, così da garantire, veridicità statistica ai nostri risultati. Come nel caso precedente, pur avendolo verificato, per non appesantire il nostro trattato, abbiamo deciso di non riportare le tabelle raffiguranti i valori degli intervalli di confidenza.
Dalla simulazione sottostante le suddette condizioni abbiamo ottenuto i seguenti valori di $AvResponseTime_c$, suddivisi nelle seguenti tabelle:

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
1,05	3,791620745	3,760166607	3,684325534	3,692665006	3,710436851
1,2	3,702283948	3,670464466	3,608880343	3,618236007	3,63320896
1,4	3,625834813	3,604151379	3,552164285	3,552139709	3,570051118
1,6	3,578235561	3,561499953	3,506985255	3,509785996	3,53173591
1,8	3,540988807	3,52841527	3,48329996	3,48082457	3,504303445

Tabella 27: AvResponseTime A Case Study 5

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
1,05	1,624957707	1,620192185	1,578203245	1,577943858	1,629346697
1,2	1,576691996	1,571311112	1,526092531	1,526205813	1,577630989
1,4	1,534593268	1,531599259	1,479967003	1,479759762	1,533077329
1,6	1,507608944	1,506718651	1,449904035	1,44994901	1,504681707
1,8	1,489358152	1,487046381	1,427203611	1,429149922	1,48693405

Tabella 28: AvResponseTime B Case Study 5

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
1,05	1,961964058	2,011987453	1,945624061	1,933322182	2,006191934
1,2	1,824790182	1,882839913	1,816520734	1,808906804	1,877096963
1,4	1,728488999	1,770344404	1,704745398	1,699131907	1,766689391
1,6	1,66872209	1,700969106	1,630928779	1,626478826	1,695455257
1,8	1,624926828	1,651673925	1,579840245	1,57853726	1,646821739

Tabella 29: AvResponseTime C Case Study 5

Da una prima analisi di queste tre tabelle, è possibile notare come gli algoritmi Norma Infinito e Norma Quadratica risultino, in termini prestazionali, migliori rispetto agli altri tre. Al fine di validare in maniera definitiva queste nostre considerazioni abbiamo deciso di costruire tre tabelle, una per ogni classe di Job, che raffigurano il guadagno percentuale dei tre algoritmi da noi costruiti rispetto a quelli già esistenti in letteratura. Ogni elemento delle Table seguenti rappresenteranno un Gain (in percentuale) che si avrà scegliendo di

adottare un algoritmo rispetto ad un altro. Per fare ciò abbiamo utilizzato la seguente formula

$$GainPerc_{ArrTime,AlgA,AlgB} = \frac{AvResponseTime_{ArrTime,AlgB} - AvResponseTime_{ArrTime,AlgA}}{AvResponseTime_{ArrTime,AlgB}}$$

dove

- $AlgA = \{NormaInfinito,\ NormaQuadratica,\ NormaComplementare\}$
- $AlgB = \{JoinTheShortestQueue,\ JoinTheShortestQueueOfClass\}$
- $ArrTime = \{1.05sec,\ 1.2sec,\ 1.4sec,\ 1.6sec,\ 1.8sec\}$

In questo modo abbiamo ottenuto le seguenti tabelle:

ShortestQ			ShortestQofC		
NormaInf	NormaQuad	NormaComp	NormaInf	NormaQuad	NormaComp
2,8298%	2,6099%	2,1411%	2,0170%	1,7952%	1,3225%
2,5229%	2,2702%	1,8657%	1,6778%	1,4229%	1,0150%
2,0318%	2,0325%	1,5385%	1,4424%	1,4431%	0,9461%
1,9912%	1,9129%	1,2995%	1,5307%	1,4520%	0,8357%
1,6292%	1,6991%	1,0360%	1,2786%	1,3488%	0,6834%

Tabella 30: GainPerc A

ShortestQ			ShortestQofC		
NormaInf	NormaQuad	NormaComp	NormaInf	NormaQuad	NormaComp
2,8773%	2,8932%	-0,2701%	2,5916%	2,6076%	-0,5650%
3,2092%	3,2020%	-0,0596%	2,8778%	2,8706%	-0,4022%
3,5597%	3,5732%	0,0988%	3,3711%	3,3847%	-0,0965%
3,8276%	3,8246%	0,1942%	3,7708%	3,7678%	0,1352%
4,1732%	4,0426%	0,1628%	4,0243%	3,8934%	0,0076%

Tabella 31: GainPerc B

ShortestQ			ShortestQofC		
NormaInf	NormaQuad	NormaComp	NormaInf	NormaQuad	NormaComp
0,8328%	1,4599%	-2,2543%	3,2984%	3,9098%	0,2880%
0,4532%	0,8704%	-2,8665%	3,5223%	3,9267%	0,3050%
1,3737%	1,6984%	-2,2100%	3,7054%	4,0225%	0,2065%
2,2648%	2,5315%	-1,6020%	4,1177%	4,3793%	0,3242%
2,7747%	2,8549%	-1,3474%	4,3491%	4,4280%	0,2938%

Tabella 32: GainPerc C

Come vediamo, gli algoritmi Norma Infinito e Norma Quadratica ci permettono un guadagno che varia a seconda della classe del Job in ingresso, ma che aumenta al crescere dell'Arrival Time. Questo è dovuto al fatto che in maniera ad esso collegato, diminuisce anche il tempo di attesa in coda dei vari Job, che vengono generati con una frequenza minore. La Norma Compatibilità purtroppo non ha risposto alle aspettative e soprattutto per Job di classe C, ottiene un guadagno negativo ovvero una perdita in termini di prestazioni. Al fine di ottenere una visione più generale, abbiamo deciso di costruire anche la tabella avente come elementi gli $AvResponseTimeWeight$ Questo vuol dire che non verranno analizzate le prestazioni suddivise per classe, ma verrà visualizzato il Response Time generico, e legato alle varie tipologie di Job solamente dalla possibilità di appartenervi. I valori ottenuti sono i seguenti:

	ShortestQ	ShortestQofC	NormaInf	NormaQuad	NormaComp
1,05	2,15179222	2,16572565	2,109653948	2,107501585	2,158618299
1,2	2,076239842	2,084600423	2,029778554	2,029422149	2,078586375
1,4	2,011010297	2,017733226	1,961839978	1,960047395	2,010555705
1,6	1,970068211	1,975950048	1,915627702	1,914875352	1,967324613
1,8	1,940354885	1,944708422	1,884213871	1,884301053	1,938374236

Tabella 33: AvResponseTimeWeight Case Study 5

ai cui sono legati i seguenti Gain percentuali

ShortestQ			ShortestQofC		
NormaInf	NormaQuad	NormaComp	NormaInf	NormaQuad	NormaComp
2,3033%	2,4030%	0,0358%	2,5890%	2,6884%	0,3282%
2,2378%	2,2549%	-0,1130%	2,6299%	2,6469%	0,2885%
2,4451%	2,5342%	0,0226%	2,7701%	2,8589%	0,3557%
2,7634%	2,8016%	0,1393%	3,0528%	3,0909%	0,4365%
2,8933%	2,8888%	0,1021%	3,1107%	3,1062%	0,3257%

Tabella 34: GainPerc Weight

I valori contenuti nella Table 34 avvalorano in maniera profonda la nostra tesi. Essi rappresentano un riassunto, ben costruito, di quello che è il quadro generale della simulazione e verranno sfruttati nel prossimo capitolo al fine di ricavare conclusioni sulla bontà del nostro operato.

6 CONCLUSIONI E POSSIBILI SVILUPPI

Nella prima parte di questo capitolo, grazie all'analisi prestazionale effettuata precedentemente e ai risultati che ne sono conseguiti, trarremo conclusioni sulla bontà dei nostri risultati e sull'effettivo miglioramento introdotto dagli algoritmi da noi creati. Nella seconda e ultima parte poi, proporremo diversi spunti, che in futuro, se sviluppati, crediamo possano portare ulteriori miglioramenti alla soluzione del "problema" di Load Balancing su macchine virtuali in ambiente Cloud.

6.1 CONCLUSIONI

Come detto il nostro obiettivo ultimo è stato quello di migliorare, in termini di $AvResponseTime$, le prestazioni di un modello rappresentante il problema del Load Balancing su macchine virtuali in ambiente Cloud. Nei casi di studio precedentemente analizzati, i risultati ottenuti sono molto incoraggianti da questo punto di vista. In quattro case study su cinque, gli algoritmi Norma Infinito e Norma Quadratica sono risultati i migliori, confermando di fatto, la bontà del nostro progetto. Il restante algoritmo da noi creato, norma Compatibilità, risulta avere risultati, in molti casi, simili ai meccanismi di instradamento già presenti in letteratura e quindi non si è rivelato all'altezza delle aspettative. Questo probabilmente è dovuto al fatto che l'indice di compatibilità, alla base del suo funzionamento, ha un peso specifico troppo basso all'interno del calcolo della macchina virtuale a cui indirizzare le richieste in ingresso. Proprio per questo motivo concludiamo che non possa essere un valido sostituto dei meccanismi di instradamento già presenti in letteratura e per questo non verrà più considerato nel proseguo del capitolo.
Dall'analisi del quinto case study poi, siamo in grado di dire che il guadagno percentuale introdotto dall'utilizzo dei nostri algoritmi, dipende fortemente dalle caratteristiche prestazionali dei vari componenti e dal valore dell'Arrival Time.
Tanto più le caratteristiche delle macchine sono "lontane" dalla condizione di riferimento ($\alpha_{k,m} = 1 \forall k \epsilon K, \forall m \epsilon M$ con $K = \{CPU, IO, DISK\}$ e $M = \{VM1, VM2, VM3\}$), tanto più il guadagno percentuale sarà maggiore e il sistema otterrà beneficio ad adottare uno i nostri algoritmi. Allo stesso modo, all'aumentare dell'Arrival Time, il guadagno percentuale aumenta e il sistema ne trae beneficio.

Queste ultime due considerazioni sono dovute al fatto che per costruzione, i nostri due algoritmi, basano la loro scelta di instradamento sul confronto dei possibili carichi futuri di lavoro, dipendenti dal service time, dei vari Job, lungo i componenti. Tanto più quest'ultima quantità sarà indipendente dai tempi di coda, tanto più i nostri algoritmi risulteranno migliori rispetto a quelli già esistenti e i guadagni percentuali del sistema aumenteranno.
Per questo motivo possiamo concludere che, considerando la crescita di capacità computazionale richiesta praticamente nulla, il sistema otterrà beneficio nell'adottare, come politica di intradamento alla risoluzione del problema Load Balancing per macchine virtuali in ambiente Cloud, un algoritmo da noi creato.
Quale scegliere tra l'algoritmo Norma Infinito e Norma Quadratica? Possiamo concludere che, a livello di prestazioni, o meglio di $AvResponseTime$, questi due meccanismi sono molto simili e trovare un vero e proprio vincitore è molto difficile. Volendo però, la Norma Quadratica ha un vantaggio dal punto di vista della "comprensione" delle caratteristiche appartenenti al sistema. Questo è dovuto al fatto che, per sua natura, quando viene calcolato l'indice di Norma alla base di questo algoritmo, esso è funzione di tutti i valori di $S_{c,k}$, a differenza del suo "rivale" che considera solamente il loro massimo.
Per questa serie di motivi, possiamo concludere che l'algoritmo Norma Quadratica sia la miglior soluzione, a pari livello di complessità computazionale, per il problema di Load Balancing su macchine virtuali in ambiente Cloud.

6.2 POSSIBILI SVILUPPI FUTURI

In questa paragrafo ci occuperemo delle possibili migliorie che possono derivare dall'approfondimento di alcuni argomenti trattati in questa tesi. Il primo aspetto che potrebbe essere sviluppato riguarda una caratteristiche delle prove effettuate per caratterizzare i nostri casi di studio, ossia la numerosità del campione. Come detto, tanto più n sarà elevato, tanto più saremo sicuri di ottenere risultati accurati e precisi. Da questo punto di vista, le prove da noi realizzate, potrebbero essere ripetute con una n molto più elevata (con $n = 100$ o superiori) per verificare che tali risultati siano effettivamente corretti e rappresentativi della realtà.
Dal punto di vista progettuale, abbiamo definito come ipotesi iniziale necessaria, la condizione che i collegamenti tra i vari moduli del modello siano sempre attivi e non soggetti a guasti e/o errori. Un possibile sviluppo potrebbe essere quello di far cadere tale ipotesi e studiare come il sistema reagirebbe a questa tipologia di problematiche. Potrebbe, per esempio, essere utile creare un meccanismo che, grazie ad una richiesta al Cloud Provider, attivi una macchina virtuale "complementare" che vada a sopperire alla VM resa "irraggiungibile" dalla rottura/guasto di uno dei collegamenti presenti.
Sempre dal punto di vista progettuale, poi, un'altra ipotesi iniziale potrebbe essere messa in discussione, ossia la condizione che tutte le code del sistema siano dotate di capacità infinità (pensiamo soprattutto al router). A tal proposito, similmente alla soluzione precedente, potrebbe essere utile duplicare tale elemento, grazie ad una richiesta di servizio al Cloud Provider, in modo che, in caso di aumento improvviso delle richieste e conseguente overflow, la copia del router di cui sopra, si attivi e parte dei Job in ingresso vengano ad esso dirottati per l'instradamento. Così facendo viene virtualmente raddoppiata la capacità del sistema e questo potrebbe risultare sufficiente ad ovviare il problema sopra descritto.
Per quanto riguarda il router poi, nel nostro progetto abbiamo trascurato una variabile temporale che nella realtà, soprattutto se il numero di richieste è molto elevato, potrebbe incidere sull'$AvResponseTime$ in maniera considerevole, ovvero il tempo di routing. Ogni router infatti, per decidere a quale macchina virtuale indirizzare il Job in Input, impiega un determinato lasso di tempo, che nel caso di un numero di richieste in coda basso può essere trascurato, ma in

caso contrario si rivela un fattore determinante. Per questo motivo, potrebbe essere sensato, nel calcolo del tempo di risposta ad una determinata richiesta, introdurre tale variabile e studiarne il comportamento al fine di migliorarne le prestazioni.
Come ultima considerazione, un possibile sviluppo futuro, potrebbe riguardare la distribuzione del Service Time e dell'Arrival Time. Come precedentemente spiegato, nei capitoli precedenti, abbiamo assunto come ipotesi iniziali che tali variabili siano rappresentate da una distribuzione esponenziale che ci restitusce, dato un numero in ingresso, un valore non deterministico (ovvero non prevedibile a priori) rappresentante il tempo di servizio e arrivo rispettivamente. Nel caso cui, fosse sostituita tale distribuzione con una dall'esito deterministico, allora potremmo calcolare, al momento della "nascita" di ogni richiesta, l'istante esatto in cui essa verrà terminata. Questo vorrebbe dire che il parametro Demand di ogni Job non assumerebbe più un valore astratto, ma bensi un valore reale. Per poter realizzare tutto ciò occorerebbe conoscere l'esatto stato del sistema in ogni momento (ad esempio il carico RealTime di ogni componente delle macchine virtuali), il che richiederebbe un incremento notevole a livello computazionale del sistema, dovuto all'aumento di complessità del problema. Lo studio della variazione delle prestazioni e di come possa essere realizzato ed implementato tale modello potrebbe essere molto interessante e potrebbe dare un importante contributo scientifico in questo campo.

7 REFERENCES

[1] http://www.vmware.com/it/cloud-computing/

[2] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee David, A. Patterson, Ariel Rabkin, Ion Stoica and Matei Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing", 2009 , Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[3]C. Hfer and G. Karagiannis, "Cloud computing services: taxonomy and comparison", in Journal of Internet Services and Applications, vol. 2, 2011

[4]R.P. Mahowald, Worldwide Software as a Service, Forecast: "Software Will never be the same" (2010–2014), in, IDC, 2010

[5]Google Apps. Available: http://www.google.com/intl/it/enterprise/apps/business/#utm_campaign=it&utm_source=it-ha-emea-it-sk&utm_medium=ha&utm_term=%2Bapps

[6]George Reese, "Cloud computing. Architettura, infrastrutture, applicazioni", 2010

[7]Ospit@ Virtuale. Available: http://www.ospitavirtuale.it/leggi/caratteristiche-del-servizio.pdf

[8]Studio Nextpolra con Microsoft, Available: http://www.microsoft.com/italy/newscenter/Default2.aspx?id=502

[9]Qi Zhang, Lu Cheng, Raouf Boutaba, "Cloud computing: state-of-the-art and research challenges", 2010

[10]K.D. Devine, E.G. Boman, R.T. Heaphy, B.A Hendrickson, J.D. Teresco, J. Faik, J.E. Flaherty, L.G. Gervasio, "New challenges in dynamic load balancing", in Numerical Mathematics, 52 (2005)

[11]Amazon Inc. Amazon Elastic Computing Cloud (EC2). Available: http://aws.amazon.com/ec2/

[12]Marios D. Dikaiakos and George Pallis, Dimitrios Katsaros, Pankaj Mehra, Athena Vakali. “Cloud Computing Distributed Internet Computing for IT and Scientific Research”, in IEEE Internet Computing, 2009

[13]Stewart Robinson, “Simulation: The Practice of Model Development and Use”, 2004

[14]Di Febbraro A., Giua A. ,“Sistemi ad Eventi Discreti”, 2002

[15]OMNeT++ homepage [online], Available: http://www.omnetpp.org

[16]Bernhard Hechenleitner and Karl Entacher, ”A simple OMNeT++ queuing experiment using different random number generators”, 2002

[17]András Varga, “OMNeT++ Discrete Event Simulation System Version 3.2 User Manual”, 2005

[18] Jinhua Hu, Jianhua Gu, Guofei Sun and Tianhai Zhao, ”A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment”, 3rd International Symposium on Parallel Architectures, Algorithms and Programming, 2010

[19]P. Cicotti, M. Taufer, A.A. Chien, Dgmonitor, “A performance monitoring tool for sandbox-based desktop grid platforms”, in The Journal of Supercomputing nr 34, 2005

[20] Luiz F. Bittencourt, Edmundo R. M. Madeira and NelsonL. S. da Fonseca, “Scheduling in Hybrid Clouds”, in IEEE 2012

[21] Shuping Ruan, Tao Jiang, Cheng Wenfang. “Hypothesis Test of Parameter in Exponential Distribution”, in 11th International Symposium on Distributed Computing and Applications to Business, Engineering & Science, 2012

[22]Edward D. Lazowska, John Zahorjan, G. Scott Graham and Kenneth C. Sevcik, “Quantitative System Performance. Computer System Analysis Using Queueing Network Models”, 1984

[23]Yongzhi Wang, Tom Z. J. Fu and Dah-Ming Chiu, "Analysis of Load Balancing Algorithms in P2P Streaming", in Allerton Conference, 2008

[24] V. Gupta, M. Harchol-Balter, K. Sigman, and W. Whitt, "Analysis of join-the-shortest-queue routing for web server farms" in Perforance Evaluation nr 64, 2007