POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Tecniche probabilistiche di dimensionamento di cloud datacenters e di decisione per l'allocazione dinamica di macchine virtuali

Relatrice: Prof.ssa Raffaela Mirandola
Correlatore: Diego Pérez
Correlatore: Andrea Montoli

Tesi di Laurea di:
Federico Monterisi, matricola 787441

Anno Accademico 2013-2014

Ai miei nonni

Indice

Ringraziamenti **3**

Introduzione **7**

1 Architettura del sistema **13**
1.1 Il cloud computing e i suoi servizi 13
1.2 Panoramica di HP CloudSystem Matrix 14
1.3 VMWare vSphere 5 16
1.4 Lo scheduler della CPU in VMWare ESXi 17
1.4.1 Scheduler in stato di over-commit 21
1.4.2 Indicatori di performance 25
1.5 Generalizzazione del modello a stati 27

2 Test e prime osservazioni **29**
2.1 Strumenti e ambiente di test 30
2.1.1 Macchine virtuali esistenti 31
2.2 Test per il livello di saturazione 32
2.2.1 Macchine virtuali con 1 *vCPU* 32
2.2.2 Macchine virtuali con 2 *vCPU* 33
2.2.3 Macchine virtuali con 3 *vCPU* 35
2.2.4 Macchine virtuali con 4 *vCPU* 35
2.3 Test per la fase di regime normale 37
2.3.1 Fase di regime normale con 4 macchine virtuali 39
2.3.2 Validazione della fase di regime normale e conclusioni 43

3 Metodi per la pianificazione della capacità **49**
3.1 Panoramica dell'algoritmo 50
3.2 Capacity Planner 53
3.3 Demand Estimator 55
3.3.1 Calcolo dell'istogramma della *demand* 58
3.3.2 Stima della *demand* 60

3.4 Indici del livello di servizio 61
3.4.1 Semplificazione dell'indice di *Response Time* 63
3.4.2 Livello di servizio aggregato 65

4 Metodi per l'allocazione dinamica di macchine virtuali 71
4.1 Metodo standard 72
4.2 Metodo con livello di *Reservation* 74
4.3 Just in Time Activation 75
4.4 Simulazioni di verifica 80
4.4.1 Metodo di simulazione 80
4.4.2 Simulazione con metodo standard 81
4.4.3 Analisi scenario con livello di *Reservation* 91
4.4.4 Impatto della tecnologia *JiTA* 93
4.5 Conclusioni 97

5 Conclusioni e sviluppi futuri 101
5.1 Conclusioni 101
5.2 Sviluppi futuri 102

Elenco delle figure 105

Elenco delle tabelle 107

Bibliografia 109

Ringraziamenti

In questa sezione vorrei ringraziare sinceramente tutte quelle persone che in qualche modo hanno contribuito e mi hanno accompagnato nella mia crescita personale, formativa e professionale. Un percorso lungo, che ha richiesto l'impegno di molte persone.

Ricordo ancora la tremenda fatica da affrontare all'età di 10 anni quando la maestra Cristina mi chiedeva di scrivere il "testo" sulla gita di classe. Ecco, forse questa tesi non sarà qualcosa che la coinvolgerà così emotivamente, ma spero che, leggendola non trovi troppi errori di ortografia.

Ricordo con piacere il primo anno di scuola media, trascorso con la professoressa Vaccarezza. Ricordo in particolare quando cercò di trasmetterci l'importanza di un "metodo" di studio. Ancora non sapevo quanto la parola "metodo" sarebbe diventata importante nel mio percorso.

Di quegli anni ricordo anche una lezione di geometria con la professoressa Filannino: dopo averci chiesto di portare un filo di spago da casa, ci fece disegnare un cerchio sul quaderno, poi ci chiese di tagliare il filo dell'esatta lunghezza della circonferenza. Quindi ci chiese di ottenere dal filo, tanti più fili possibili di lunghezza uguale al diametro del cerchio. Ne ottenemmo 3 e un po', o meglio 3,1415...

I 5 anni di liceo, sono stati lunghi alle volte interminabili, ma certamente indispensabili a farmi crescere e mettermi sulla strada giusta per affrontare in modo sereno il mondo dell'università. Un grazie particolare lo voglio rivolgere al professor Cervesato, insegnate di Fisica e Matematica. Oltre alla didattica canonica, un giorno ci illustrò uno schema semplicissimo su come organizzare le proprie attività. Se suddivise tra necessarie e non necessarie e tra urgenti e non urgenti, vanno eseguite in questo ordine:

1. attività necessarie urgenti
2. attività non necessarie urgenti
3. attività necessarie non urgenti
4. attività non necessarie non urgenti

Quello che ho letto in questo messaggio è che tutte le attività, anche quelle che alle volte ci sembrano meno importanti (o meno necessarie), che siano di semplice svago oppure no, ad un certo punto è giusto che ottengano il loro spazio e vengano portate a termine. Pensandoci bene, forse la laurea magistrale avrei dovuto catalogarla nelle attività necessarie e urgenti un po' prima!

Negli anni dell'università ho avuto l'opportunità di conoscere molte persone competenti e appassionate al loro lavoro. A tutte queste persone devo un grazie per avermi sempre spinto a dare di più e ad essere convinto nelle scelte fatte (rifiuto o accetto un voto? riparto subito con gli esami o provo un'esperienza lavorativa?).

Un grazie particolare lo rivolgo alla professoressa Mirandola, mia relatrice, e a Diego, mio correlatore, per avermi dato la possibilità di lavorare assieme a loro e per avermi offerto la possibilità di realizzare questo progetto di tesi, supportato dalla professionalità e competenza di Andrea Montoli.

Ringrazio tutti i colleghi di HP Italia che ho avuto l'opportunità di conoscere durante il mio periodo di stage. Ringrazio soprattutto Marco che mi ha accompagnato durante l'ingresso nella realtà lavorativa e che mi ha permesso di integrarmi da subito nel gruppo di quelli di Cernova.

In questo mio breve excursus scolastico e lavorativo, volevo lasciar spazio anche a quelle persone con cui ho condiviso gran parte del tempo tra i banchi di scuola e di università. Grazie a Lorenzo per tutte le ore di scuola elementare, media e liceo passate assieme e grazie a Tommaso per gli anni di liceo. Grazie ai compagni di università e di progetto: Marco, Roberto (aka Gerry), Claudia, Serena, Andrea (Mambretti), Andrea (Martinelli), Andrea (Molica), Francesco (Filipazzi) Francesco (Grazioli), Carlo, Gregorio e Alberto.

No Andrea Gandini, non ti ho dimenticato dall'elenco. È solo che volevo ringraziarti perché se non fosse per diversi tuoi appunti e per le diverse ore passate in università con te a studiare, forse a quest'ora sarei ancora indeciso sugli esami da mettere (o togliere) nel piano di studi!

Grazie ad una persona che considero un po' come un compagno di corso, forse un po' grande, ed un po' come insegnante! Grazie Edoardo, per avermi fatto conoscere un mondo che solo adesso mi accorgo essere ancora così vasto!

Queste ultime righe le dedico alle persone più vicine a me, che di giorno in giorno mi rendono felicemente consapevole della fortuna che ho avendole accanto. Grazie mamma, grazie papà per la fiducia che avete riposto in me e per quella non riposta, alle volte è giusto sapersela (ri)guadagnare. Grazie perché questo traguardo è anche vostro.

Grazie Stefano, perché mi permetti di rivivere le sfide affrontate nel mio/nostro percorso scolastico e universitario e di capire quanto valga l'obiettivo raggiunto! E ti assicuro, ne vale la pena!

Grazie Erica, per avere condiviso con me i successi e per aver riportato, di tanto in tanto, la giusta dimensione agli insuccessi. Ci siamo conosciuti quando credevo di essere un uomo, ora forse lo sono diventato, anzi diciamo che lo sto diventando. Diciamo anche che non c'è una laurea che abilita alla professione, ma in ogni caso da ora potremo conoscere ed esplorare la vita senza più l'ansia delle scadenze degli appelli d'esame!

Grazie a tutti!

Federico

Introduzione

L'avvento del cloud computing è stato un evento che ha condizionato in maniera significativa l'evoluzione del mondo tecnologico negli ultimi anni. Il cloud computing, nel senso più lato, individua l'insieme delle tecnologie che permettono la realizzazione di architetture IT attraverso le quali vengono offerti servizi di vario tipo attraverso la rete. I servizi sono distinti a seconda del livello di astrazione del software.

Partendo dal livello più alto, è possibile individuare servizi di tipo SaaS, o Software as a Service, che offrono, sostanzialmente, applicazioni fruibili da remoto senza, quindi, la necessità di installazione. Spesso questi software sono direttamente accessibili da browser il che maschera in maniera quasi completa l'architettura IT sottostante e la piattaforma su cui poggia il software stesso.

Al livello sottostante è possibile trovare i servizi cosiddetti PaaS, Platform as a Service. Questo genere di servizi offrono al cliente un ambiente di sviluppo capace di velocizzare la realizzazione del software, astraendolo dal sistema operativo sottostante. Non solo, il software, in questo contesto, può essere anche distribuito, il tutto in totale trasparenza rispetto al cliente. Google App Engine o Microsoft Azure sono solo due degli esempi possibili di questo tipo di servizi.

A livello più basso esistono i servizi cosiddetti IaaS, Infrastructure as a Service, che offrono architetture IT virtuali su cui è possibile installare e gestire in autonomia il proprio sistema operativo.

Il fattore comune delle tre tipologie di servizi, fattore che ha determinato anche il crescente utilizzo del cloud computing in ambito industriale, è la possibilità di utilizzare il servizio offerto senza la necessità di acquistare in maniera permanente licenze software o hardware fisico. Si ha quindi la possibilità di sfruttare l'offerta del provider per il tempo necessario al suo utilizzo.

Se da lato cliente questo porta notevoli vantaggi, lato provider vengono richieste una serie di soluzioni in grado di dimensionare "correttamente"

l'hardware necessario (i.e. il provider deve trovare il giusto compromesso tra capacità delle risorse e performance offerte al cliente).

Obiettivo di questo elaborato è affrontare l'esigenza da parte del provider di massimizzare l'utilizzo delle risorse, mantenendo sempre un occhio di riguardo alle performance misurate dal cliente.

In questo scenario i due attori principali sono il provider, che offre il proprio portafoglio di servizi, e il cliente che fruisce dei servizi e che pretende una certa qualità del servizio. Per entrambi gli attori è importante l'impatto economico che il servizio comporta. Infatti se per il cliente è rilevante l'abbattimento dei costi di gestione del servizio (i.e. in un servizio IaaS il cliente non si occupa dell'hardware fisico, in un servizio PaaS non ha la necessità di gestire gli aggiornamenti del sistema operativo, etc.), per il provider è importante non sovradimensionare la capacità complessiva che viene offerta.

Il rischio da parte del provider è, quindi, duplice:

- se il datacenter è sovradimensionato, molte delle sue risorse che potrebbero essere impiegate vengono, di fatto, sprecate

- se il datacenter è sottodimensionato, il cliente può avvertire cali di performance che violino il contratto stipulato con il provider

Questo elaborato si sviluppa dal lato provider: avendo la visione completa della struttura fisica su cui poggia il sistema di virtualizzazione, si vuole trovare un metodo capace di prevedere la necessità di installare nuovo hardware nel datacenter e, nel caso, di dimensionarlo correttamente. Un'analisi di questo tipo è finalizzata a report periodici (verosimilmente con cadenza semestrale) che monitorino lo stato del datacenter per valutare l'eventuale estensione dello stesso.

In questo contesto assume anche una notevole importanza la necessità di riconoscere a run-time se un host, macchina fisica cui vengono assegnate le macchine virtuali, è in grado di allocare una nuova macchina virtuale o meno. Cioè, una volta dimensionato il datacenter e avendo a disposizione una serie di host, è importante capire al momento della ricezione dell'ordine, se la macchina virtuale può essere assegnata o meno ad un host e con quali effort.

Nell'ottica di proporre al provider una soluzione che fosse in grado di rispondere ad entrambe le esigenze, senza entrare nel merito della tipologia di utilizzo che il cliente fa con le macchine virtuali da lui richieste, si è arrivati alla definizione di due metodi:

Capacity Planning tecnica per dimensionare il datacenter facendo affidamento sui dati di utilizzo delle macchine attualmente allocate e sulla previsione di aumento del numero di macchine virtuali che richiedono l'allocazione;

Dynamic Decision tecnica per riconoscere a run-time se una certa macchina fisica è in grado di allocare nuove macchine virtuali.

Entrando più nel dettaglio, la tecnica di Capacity Planning sfrutta essenzialmente tre informazioni disponibili al provider:

- configurazione degli host che compongono il datacenter
- utilizzo e configurazione delle macchine virtuali attualmente allocate
- incremento previsto delle macchine virtuali

In questo modo è possibile realizzare un modello che, utilizzando informazioni dinamiche, quali l'utilizzo delle macchine virtuali, con informazioni statiche, quali la configurazione degli host e delle macchine virtuali, sia in grado di quantificare l'hardware necessario a garantire un certo livello di performance.

Proprio a questo fine vengono introdotti nel modello due indici di performance che vengono poi riutilizzati anche nella tecnica di Dynamic Decision.

La Dynamic Decision è una soluzione che si rileva utile al provider nel momento in cui deve stabilire se un host è in grado di ospitare una nuova macchina virtuale. In questo contesto vengono sfruttati gli stessi indici introdotti nella tecnica precedente, stimati sulla base di informazioni diverse:

- configurazione del singolo host
- utilizzo e configurazione delle macchine virtuali attualmente allocate
- utilizzo stimato della nuova macchina virtuale

Se, a seguito dell'analisi, gli indici di performance risultano accettabili, le risorse dell'host vengono ritenute sufficienti per la virtualizzazione della nuova macchina virtuale.

Dato l'interesse mostrato da Hewlett-Packard Italia per la tecnica di Dynamic Decision, si è voluto approfondire ed estendere la soluzione a tre scenari:

Scenario Standard le macchine virtuali allocate su un singolo host hanno la garanzia di raggiungere dei livelli di performance dipendenti solamente del loro utilizzo pregresso;

Scenario con *Reservation* le macchine virtuali oltre alla garanzia sui livelli di performance, ottengono la garanzia su un livello minimo di risorse disponibili;

Scenario con *Just in Time Activation* l'utente ha la possibilità di configurare le macchine virtuali (sia a livello hardware, sia a livello applicativo) ancora prima di allocarle, questo fa sì che il suo impatto non sia immediato sull'host.

L'ultimo scenario è il frutto di una nuova tecnologia che HP sta sviluppando all'interno di una soluzione cloud rivolta al mondo enterprise. Più precisamente il *Just in Time Activation*, o semplicemente *JiTA*, mira ad offrire al cliente la possibilità di configurare la macchina virtuale senza la necessità di accenderla. Il processo di creazione e allocazione delle macchina virtuale, la sua configurazione lato software (installazione del sistema operativo e delle applicazioni) viene del tutto automatizzato solo all'effettiva richiesta di utilizzo da parte del cliente.

La collaborazione intercorsa con HP durante il lavoro di tesi, ha permesso di calare in una realtà industriale i due metodi. Comprendendo le esigenze dei provider e i limiti delle soluzioni proposte, si è potuto impostare i test per la valutazione dei modelli proposti e le simulazioni in grado di validare la teoria su cui si poggiano i due metodi.

Data un'analisi pregressa fatta dalla stessa HP, è risultato evidente come la risorsa più critica da gestire fosse la potenza di calcolo. Il livello di maturazione raggiunto dalle architetture multi-core e dai software di virtualizzazione, ha fatto sì che l'efficienza dei sistemi cloud raggiungesse livelli comparabili ai sistemi fisici standard [6], anche in situazioni di over-booking (i.e. un sistema fisico con 8 core fisici con allocati un numero di core virtuali maggiore di 8).

L'esigenza di individuare gli stati critici in cui il calo delle performance misurato dalle macchine virtuali non potesse essere più ritenuto accettabile, ha richiesto lo studio di due principali fattori:

- il funzionamento dello scheduler nel sistema cloud studiato
- la conoscenza del potenza di calcolo richiesta dalle singole macchine virtuali, o semplicemente *demand*

Concentrandosi sull'architettura messa a disposizione da HP, il Capitolo 1 presenta una panoramica del sistema cloud, soffermandosi nelle ultime sezioni sulla tecnica di scheduling adottata da VMware, hypervisor utilizzato in questa soluzione IaaS.

La seconda esigenza, la conoscenza della *demand*, nasce dalla necessità di prevedere quale carico di lavoro deve aspettarsi il provider dalle macchine virtuali attualmente allocate. È già noto da altri studi [13] che individuare la potenza di calcolo richiesta da una macchina virtuale non presenta una soluzione banale, difatti l'imprevedibilità della *demand* non è risolvibile con il solo monitoraggio dell'utilizzo effettivo della CPU.

Noto il comportamento dello scheduler e realizzato un metodo di stima per la *demand* attraverso una tecnica di *Uncertainty Management*, è stato possibile realizzare delle soluzioni da fornire al provider per gestire la potenza di calcolo del proprio datacenter.

Il resto dell'elaborato è organizzato come di seguito esposto.

Capitolo 1 Offre al lettore la panoramica del sistema cloud oggetto della tesi. Con un'impostazione top-down affronta dapprima l'architettura della soluzione HP CloudSystem Matrix, quindi si occupa delle tecnologie utilizzate soffermandosi sull'hypervisor di riferimento: VMware. Viene infine analizzato la tecnica di scheduling adottata dall'hypervisor e gli indicatori di performance monitorabili.

Capitolo 2 Descrive i test effettuati per la definizione di un modello capace di stimare il degrado di performance all'aumentare della richiesta di risorse. Analizzando in primo luogo i casi in cui la *demand* delle macchine virtuali mantiene dei picchi costanti al 100%, il capitolo mostra come il modello sia stato gradualmente perfezionato fino a raggiungere un buon livello di affidabilità.

Capitolo 3 Illustra la soluzione adottata per la tecnica di Capacity Planning che si traduce in un algoritmo capace di stimare l'hardware necessario per sopperire all'aumento di richieste di allocazione. La composizione della soluzione viene mostrata attraverso uno sviluppo top-down in cui, dopo aver mostrato il funzionamento dell'algoritmo, viene proposta una soluzione per la stima della *demand*. In ultimo vengono introdotti gli indici del livello di servizio che permettono di quantificare il grado di performance offerto al cliente.

Capitolo 4 Mostra la tecnica di Dynamic Decision che viene estesa in maniera progressiva: a partire dallo scenario standard, si passa allo scenario con *Reservation* e infine allo scenario con *JiTA*. Il capitolo espone anche le simulazioni effettuate con l'obiettivo di misurare la validità degli approcci mostrati.

Capitolo 5 Evidenzia i risultati raggiunti, le possibili applicazioni e alcuni spunti che possono estendere lo studio effettuato migliorando ulteriormente i modelli presentati.

Capitolo 1

Architettura del sistema

In questo capitolo viene delineata l'architettura cloud in cui andranno ad operare i sistemi studiati. Questa architettura è parte di un deployment industriale in produzione, realizzato da HP Italia.
Dopo una prima panoramica dell'ambiente cloud offerto dalla soluzione HP CloudSystem Matrix, viene descritta la piattaforma di VMware che gestisce i servizi di IaaS. La descrizione dell'architettura prosegue con l'analisi del funzionamento dello scheduler in VMWare ESXi 5.1 e degli indicatori di performance riguardanti le CPU virtualizzate, o semplicemente $vCPU$.

Prima di entrare nel vivo dell'architettura è doveroso comprendere il concetto di cloud computing e dei servizi che esso mette a disposizione.

1.1 Il cloud computing e i suoi servizi

Il cloud computing è una tecnologia che mira ad offrire una quota di risorse creando infrastrutture IT (es. CPU, Memoria, Disco, Applicazioni, etc.) virtuali on-demand e offrendole sotto forma di servizi. Ognuno di essi viene classificato con l'acronimo XaaS, X as a Service. I più noti sono sostanzialmente tre:

SaaS - Software as a Service sono tutti quei servizi che offrono pacchetti software pronti per l'utente finale. Si annoverano in questa categoria suite quali Google Apps (Gmail, Google Calendar, Google Drive, Google Maps, etc.) o Microft Office 360.

PaaS - Platform as a Service sono indirizzati ad un pubblico di sviluppatori e predispongono ambienti in cui è possibile creare proprie applicazioni e pubblicarle in rete. Esempi di questi servizi sono Google AppEngine o Microsoft Azure.

IaaS - Infrastructure as a Service è una tipologia di servizi rivolta ai professionisti dell'IT che hanno la necessità di avere a disposizione un ambiente virtuale con un proprio sistema operativo, una certa quantità di memoria e di potenza di calcolo oltre ovviamente allo spazio su disco.

Tutti i servizi di tipo XaaS hanno due caratteristiche basilari in comune: vengono forniti attraverso la rete, il loro utilizzo è quantificato in tempo di utilizzo o di spazio occupato. In particolare per i servizi IaaS, questo ha il vantaggio di non obbligare l'utente ad acquistare hardware, grazie alla creazione temporanea di un ambiente equivalente.

La grande flessibilità di questi sistemi e la pervasiva diffusione delle reti (pubbliche e private) ha fatto sì che la tecnologia del cloud computing si imponesse come una delle principali fonti di innovazione per le aziende.

Per rispondere ad una domanda sempre più crescente, HP ha creato una serie di soluzioni tra cui quella denominata HP CloudSystem Matrix che di seguito verrà brevemente descritta.

1.2 Panoramica di HP CloudSystem Matrix

HP CloudSystem Matrix è una soluzione rivolta alle aziende per la distribuzione di ambienti cloud ibridi e privati, creata integrando tecnologie HP esistenti quali:

HP BladeSystem - macchine fisiche utilizzate per l'infrastruttura hardware;

HP Matrix Operating Environment - software per la gestione di servizi di tipo IaaS, cuore dell'intera soluzione;

HP Cloud Service Automation for Matrix - software per l'automazione del ciclo di vita di server, fisici e virtuali, e di applicazioni.

In un tradizionale ambiente IT lo sviluppo di una nuova applicazione richiede il coinvolgimento di diverse persone che vanno tra loro coordinate. Esiste la figura dell'architetto IT che sceglie i server, le macchine virtuali, lo spazio su disco e le reti necessari all'applicazione. Il designer, invece, specifica come deve avvenire la comunicazione tra tutti i componenti e quali sono le policy e gli standard dell'ambiente. Infine gli specialisti IT utilizzano effettivamente l'applicazione sviluppata e si occupano del mantenimento dei singoli componenti.

Partendo da queste figure HP CloudSystem Matrix individua tre categorie di utenti:

amministratore può creare dei pool di risorse e monitorare il loro utilizzo;

architetto IT può progettare l'architettura (virtuale e fisica) per le applicazioni aziendali;

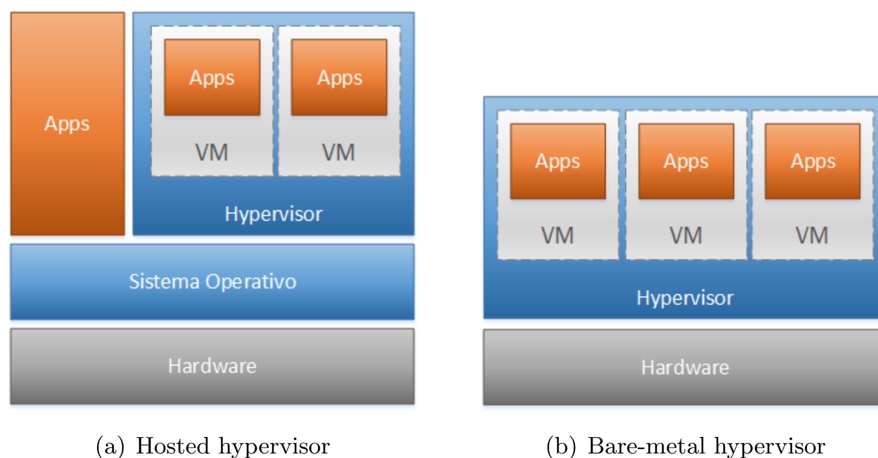
utente utilizza i servizi disponibili all'interno di un catalogo creato tramite le linee guida dettate dall'architetto.

HP CloudSystem Matrix offre un'unica piattaforma d'accesso per poter coordinare la richiesta, l'approvazione e la fornitura dei servizi. A queste funzionalità si affianca l'attività di monitoraggio e di ottimizzazione.

In una soluzione che offre servizi di tipo IaaS giocano un ruolo chiave la possibilità di stimare il dimensionamento delle risorse fisiche in fase di design e la capacità di riconoscere il migliore bilanciamento delle risorse virtuali all'interno del sistema. Un sistema ha un ottimo bilanciamento se riesce a sfruttare in maniera efficiente l'hardware fisico, senza degradare la qualità del servizio offerto. HP CloudSystem Matrix integra un software di simulazione di scenari di utilizzo che, analizzando i picchi del carico di lavoro richiesto per le varie risorse, è in grado di valutare l'impatto che avranno le architetture virtuali una volta allocate sulle diverse macchine fisiche.

HP CloudSystem Matrix fa affidamento sui log che registrano l'utilizzo delle macchine virtuali presenti nel sistema e propone come capacità ideale il picco massimo, ottenuto sommando tra di loro i valori dei log. Questo metodo, noto come peak-of-sums, garantisce una buona robustezza del sistema, smussando l'inefficienza a causa del sovradimensionamento delle risorse fisiche del metodo sum-of-peaks, in cui la capacità ideale è data dalla somma dei picchi di ogni macchina virtuale. La principale causa di sovradimensionamento è data dal fatto che i picchi di utilizzo delle macchine virtuali non si verificano obbligatoriamente nello stesso istante di tempo.

Poiché il modello che si vuole sviluppare in questo elaborato necessita della *demand* delle macchine virtuali come parametro di input è stato necessario approfondire le fondamenta su cui HP CloudSystem Matrix poggia la fornitura di servizi di tipo IaaS. La soluzione HP si interfaccia con tre principali ambienti di virtualizzazione: Microsoft Hyper-V, HP Integrity VMs e VMware. In particolare nelle ultime implementazioni si è preferito il binomio HP CloudSystem Matrix-VMware e, essendo già predisposto un ambiente di laboratorio di questo tipo su cui poter fare esperimenti, si è scelto di approfondire questa soluzione.



(a) Hosted hypervisor (b) Bare-metal hypervisor

Figura 1.1: Tipologie di hypervisor

1.3 VMWare vSphere 5

VMWare vSphere nasce con lo scopo di aggregare e rendere scalabili tutte le risorse offerte da uno o più datacenter e tradurli in servizi di tipo IaaS. L'ostacolo più grande nella fornitura di questo tipo di servizi è riuscire a separare nel modo più netto possibile il livello applicativo da quello hardware. Questo approccio permette di sfruttare in maniera più efficiente l'hardware fisico, sostituendo a più server fisici macchine virtuali ospitate da un unico server.

L'ambiente virtuale creato sui server fisici attraverso VMWare vSphere viene classificato come hypervisor di tipo *Bare-metal*. Questa tipologia di hypervisor si differenzia dagli hypervisor *Hosted* poichè, mentre quest'ultimi offrono un'infrastruttura virtuale attraverso un applicativo installato nel sistema operativo (es. VMWare Workstation, Oracle VirtualBox), essi si interfacciano direttamente con l'hardware fisico gestendo in autonomia le richieste delle macchine virtuali.

Un'idea di massima della differenza tra le due categorie di hypervisor viene data nella Figura 1.1 dove si può vedere che nel caso di hypervisor Hosted (1.1(a)), l'ambiente di virtualizzazione viene installato sopra al sistema operativo e viene affiancato da tutti gli applicativi presenti nello stesso. Nel caso di hypervisor Bare-metal(1.1(b)), invece, la macchina fisica viene predisposta per la sola virtualizzazione garantendo una maggiore efficienza nell'utilizzo dell'hardware da parte delle macchine virtualizzate.

Il punto critico dell'architettura Bare-metal è, per l'appunto, il livello software che maschera al sistema operativo delle macchine virtuali la gestione

dell'hardware. In VMWare vSphere questo compito è svolto da VMWare ESXi, pacchetto software che viene installato sulle macchine fisiche, anche dette *host*. Per *host* si intende l'insieme di risorse di memoria e di potenza di calcolo della macchina fisica. Lo spazio su disco è invece fornito da quello che nell'ambiente VMware viene definito *datastore*, contenitori logici che si possono creare su diversi tipi di storage fisici (es. dischi locali, SAN, ...).

Più host possono essere raggruppati in *cluster* creando un'entità unica capace di erogare una quantità di risorse pari, in linea teorica, alla somma degli stessi. Ogni macchina virtuale può essere assegnata ad un host o ad un cluster. In questo modo le risorse richieste verranno fornite in maniera dinamica in base all'ambiente.

L'architettura di VMWare vSphere si suddivide in tre livelli (vedi Figura 1.2):

Livello di interfaccia - Comprende tutti i moduli software che permettono sia all'utente finale (attraverso vSphere Client o vSphere Web Client), sia a software di terze parti (attraverso vSphere SDK), di collegarsi alla piattaforma.

Livello di gestione - È occupato interamente da VMWare vCenter Server. Questo riceve le istruzioni dalle interfacce ed attua le operazioni di configurazione, provisioning e gestione sull'ambiente virtualizzato.

Livello di virtualizzazione - È composto essenzialmente dai cluster e/o dagli host VMWare ESXi. Fornisce le funzionalità di calcolo, di archivio dati e di rete.

Per poter fornire uno strumento in grado di pianificare la capacità di calcolo necessaria alla creazione di un'infrastruttura per il cloud computing è stato necessario approfondire il funzionamento del livello di virtualizzazione. In particolare si è cercato di capire quali fossero gli algoritmi che governano lo scheduler della CPU e quali i parametri di misurazione utili alla creazione del modello di dimensionamento.

1.4 Lo scheduler della CPU in VMWare ESXi

Lo scheduler di VMWare ESXi interpreta ogni *vCPU* come un contesto di esecuzione che necessita di una CPU fisica, o *pCPU*. Il numero di *pCPU* dipende dall'architettura della CPU. Se dotata di tecnologia hyper-threading il numero di *pCPU* è dato dal numero dei processori logici. Se, invece, la CPU non è dotata di tecnologia di hyper-threading, o semplicemente questa non è attiva, il numero di *pCPU* è dal numero di core della CPU.

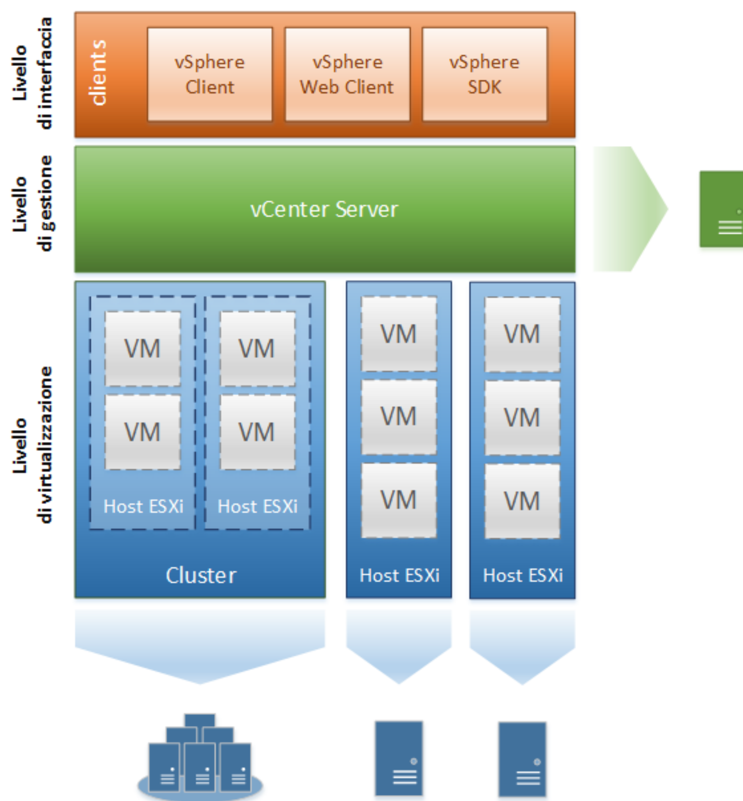


Figura 1.2: Architettura VMWare vSphere 5

Il “contesto d’esecuzione” è assimilabile al concetto di processo all’interno di un sistema operativo tradizionale ed è definito con il termine di *world*. Ogni world è legato ad una *vCPU* ed ha associato un proprio stato che è determinato dal suo grado d’esecuzione e da quello degli altri world presenti nella stessa macchina virtuale.

Gli stati in cui si può trovare un world sono essenzialmente quattro:

Run - il world ha una *pCPU* assegnata e sta sfruttando la risorsa;

Wait - il world non ha nulla da elaborare o è in attesa delle interfacce di I/O;

Ready - il world vorrebbe poter utilizzare una *pCPU*, ma al momento il sistema non può assegnargliene alcuna;

Co-stop - il world è in attesa di uno o più world per poter sincronizzare le informazioni e continuare l’elaborazione.

Mentre i primi tre sono di facile comprensione, anche perché assimilabili agli stati di *Run*, *Ready-To-Run* e *Wait*, assegnati ad un processo all’interno di un algoritmo generico Round-Robin [8], il quarto merita alcune righe di approfondimento.

Lo stato di *Co-stop* nasce dall’esigenza di parallelizzare la computazione nelle macchine virtuali multi-core. Per esempio, quando 2 *vCPU* devono eseguire istruzioni di calcolo in un ambiente virtuale è possibile che non siano effettivamente disponibili 2 unità fisiche. Per non degradare troppo le prestazioni le 2 *vCPU* possono utilizzare in maniera alternata una sola *pCPU*. Un world si trova in stato di *Co-stop* quando è stato mandato in esecuzione prima del secondo e rimane in attesa che quest’ultimo raggiunga il suo stesso livello di avanzamento.

La Figura 1.3 mostra l’avanzamento di quattro world di una stessa macchina virtuale in fase di scheduling. Ogni rettangolo rappresenta un intervallo di tempo T in cui il word esegue delle operazioni. Le frecce arancioni misurano il tempo d’avanzamento rispetto al primo world più *lento*. Lo scheduler all’istante $5T$ potrà decidere se:

- continuare ad elaborare tutti i world parallelamente
- mettere in *Co-stop* il world 4 o in *Ready* gli altri world a seconda della disponibilità di *pCPU*

L’evoluzione dello scheduler dalla versione 3 alla versione 4 ha dotato il sistema di una maggiore flessibilità consentendo un maggiore partizionamento dell’unità da eseguire [9]. Le opzioni di scheduling sopra citate sono

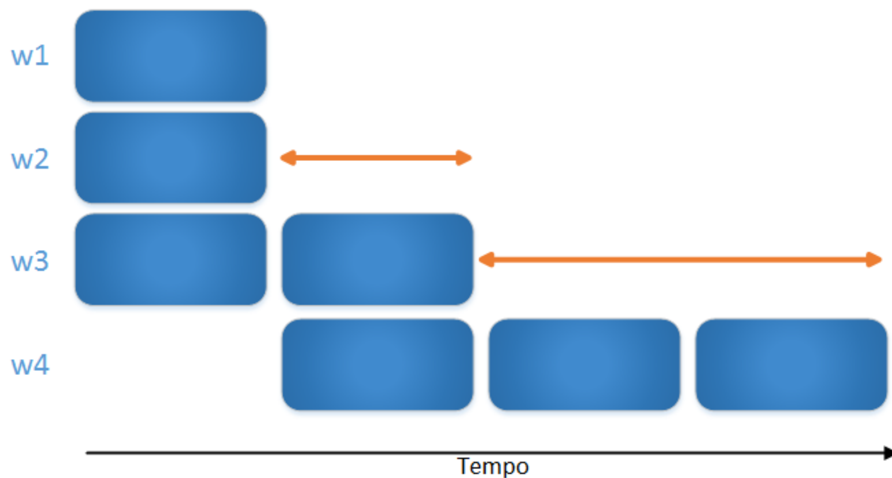


Figura 1.3: Schema d'esecuzione di 4 world disallineati.

quelle riferite dalla versione 4 in poi. In passato il sistema cercava di mandare in esecuzione tutti i world che accumulavano un ritardo *troppo* elevato. Nel caso in figura, per esempio, lo scheduler poteva scegliere di mandare in esecuzione: (w1, w2, w3, w4), (w1, w2, w3), (w1, w2, w4), (w1, w2).
Di contro dalla versione 4 si è scelti di tener conto del world più avanzato e bloccare, nell'eventualità solo quello. Nelle esempio in figura le opzioni di scheduling diventano quindi : (w1, w2, w3, w4), (w1, w2, w3), (w1, w2, w4), (w1, w2), (w1, w3), (w1, w2), (w1), (w2), (w3).

Entrambe queste soluzioni vengono identificate come implementazioni del cosiddetto Co-Scheduling *rilassato* che si contrappone al Co-Scheduling *stretto* in cui, nel momento in cui i world accumulano troppo ritardo l'uno con l'altro, tutte le *vCPU* della macchina virtuale vengono messe in stato di *Co-stop* [10]. Le *vCPU* rimarranno in questo stato finché il sistema non è in grado di mandarle tutte in esecuzione. Il Co-Scheduling stretto è stato abbandonato dalla versione 3 in favore di quello rilassato, quindi è stato migliorato nelle versioni 4 e 5. Per gli obiettivi di questa tesi è stata usata la versione 5.

Per completezza nel modello a stati che definisce il ciclo di vita di un world vengono aggiunti lo stato di *New* (world in fase di creazione) e lo stato di *Zoombie* (world in fase di rimozione).

La Figura 1.4 mostra tutti gli stati e quali sono i segnali che fanno passare un world da uno stato all'altro. Da notare che un world in stato di *Co-stop* prima di tornare in *Run* deve per forza passare prima dallo stato di *Ready*.

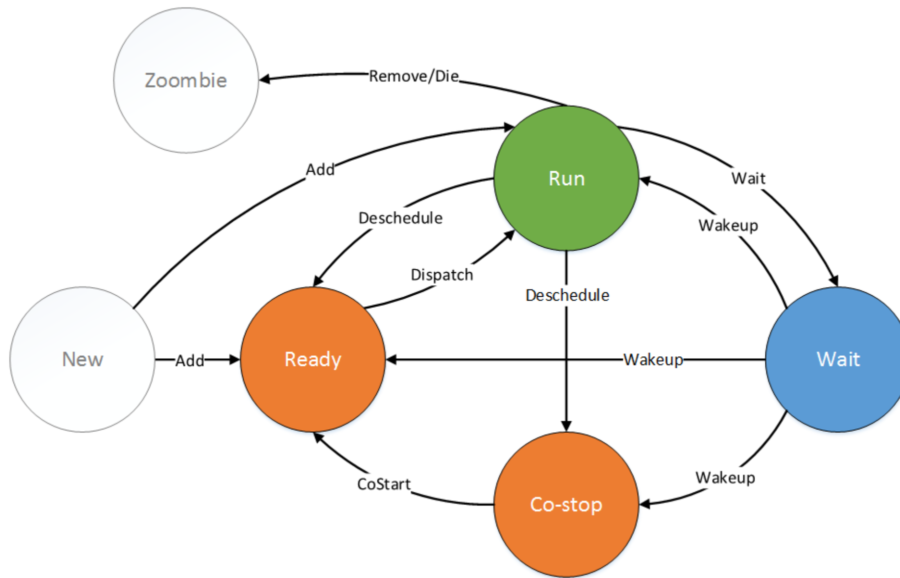


Figura 1.4: Modello a stati dei world in VMWare ESXi

1.4.1 Scheduler in stato di over-commit

Per comprendere in maniera completa il funzionamento dello scheduler di VMWare ESXi e quindi valutare gli scenari critici in cui si può trovare l'host, è necessario sapere come si comporta l'algoritmo di scheduling quando la *demand* delle macchine virtuali supera la capacità fisica dell'host.

A questo scopo VMWare ESXi implementa un algoritmo *share-based* [2][5][11] che permette di assegnare ad ogni macchina virtuale un certo numero di *quantum di tempo*, o unità di tempo (*tu*), in modo proporzionale al loro valore di *Shares*. Il concetto di *Shares* è assimilabile al concetto di priorità.
Ad esempio, quando due macchine virtuali hanno un valore di *Shares* rispettivamente di 100 e 200, nel caso in cui il sistema non sia in grado di soddisfare la richiesta di entrambe le macchine, quella con *Shares* pari a 200 riceverà due volte le risorse dell'altra macchina. In sostanza il valore di *Shares* viene utilizzato come peso nell'assegnazione delle risorse nei momenti di over-commit.

Non essendo l'algoritmo *share-based* oggetto di questa tesi, in questa sezione si vuole fornire un metodo semplice per comprendere come avviene la suddivisione della capacità di calcolo totale di un host con N^{pCPU}, numero di CPU fisiche, data una serie di macchine virtuali. Per ogni macchina v si avrà un valore di *demand* pari a $d_v^{\%}$, numero di *vCPU* pari a N_v^{vCPU} e valore di *Shares* pari a S_v. Il metodo permetterà di calcolare $a_v^{\%}$, percentuale

della capacità fisica assegnata alla macchina v.
Presupposto fondamentale è che il valore $a_v^{\%}$ sia misurato su un intervallo di tempo $t \gg tu$.

Il metodo di allocazione della capacità di calcolo si può comprendere tramite un processo iterativo composto da quattro passaggi.
Siano

$\mathcal{VM}^{host}$ l'insieme di tutte le macchine virtuali presenti nell'host;

$A^{\%}$ la capacità di calcolo a disposizione dell'host, inizialmente pari al 100%;

$a_v^{\%}(i)$ la capacità di calcolo assegnata alla macchina virtuale v all'iterazione i. Per definizione $a_v^{\%}(0) = 0$

$\mathcal{VM}$ l'insieme delle macchine virtuali che necessitano di capacità di calcolo (formalmente $v \in \mathcal{VM} \Leftrightarrow a_v^{\%} \cdot \frac{N^{pCPU}}{N_v^{vCPU}} < d_v^{\%}$);

Inoltre sia inizialmente $\mathcal{VM} \equiv \mathcal{VM}^{host}$.

Si svolgano nell'ordine le seguenti operazioni.

1. Per ogni macchina virtuale $v \in \mathcal{VM}$ viene calcolata la capacità fisica da assegnare secondo la seguente formula:

$$a_v^{\%}(i) = min(a_v^{\%}(i-1) + A^{\%} \cdot \frac{S_v}{\sum\limits_{u \in \mathcal{VM}} S_u}, d_v^{\%} \cdot \frac{N_v^{vCPU}}{N^{pCPU}})$$

2. Il valore $A^{\%}$ viene aggiornato secondo i valori di $a_v^{\%}(i)$:

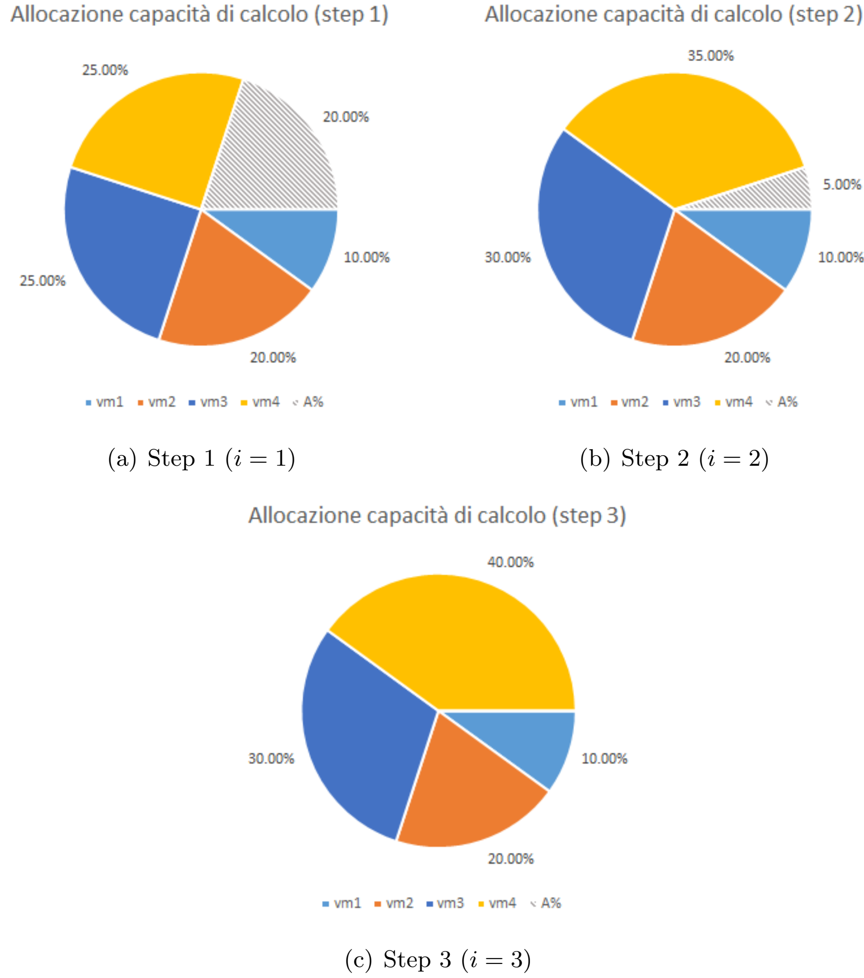
$$A^{\%} = 1 - \sum_{v \in \mathcal{VM}^{host}} a_v^{\%}(i)$$

3. Vengono rimossi dall'insieme $\mathcal{VM}$ le macchine virtuali v che vedono la loro *demand* soddisfatta, che hanno, cioè,

$$a_v^{\%} \cdot \frac{N^{pCPU}}{N_v^{vCPU}} = d_v^{\%}$$

4. Se $\mathcal{VM} = \emptyset$ o se $A^{\%} = 0\%$ l'algoritmo termina, altrimenti i viene incrementato e si ripetono le operazioni dal passaggio 1.

La Figura 1.5 mostra l'applicazione dell'algoritmo descritto dato il seguente scenario. Sia $N^{pCPU} = 4$ il numero di $pCPU$ dell'host che virtualizza quattro macchine $vm1$, $vm2$, $vm3$, $vm4$, tutte e quattro con stesso valore



(a) Step 1 ($i = 1$)

(b) Step 2 ($i = 2$)

(c) Step 3 ($i = 3$)

Figura 1.5: Esempio di suddivisione della capacità di calcolo con macchine virtuali di uguale *Shares*

di *Shares* e con $N_v^{vCPU} = 2$. Sia, infine, la *demand* delle macchine virtuali rispettivamente pari a $d_{vm1}^{\%} = 20\%$, $d_{vm2}^{\%} = 40\%$, $d_{vm3}^{\%} = 60\%$, $d_{vm4}^{\%} = 90\%$. Data la *demand* delle macchine virtuali, il loro numero di *vCPU* ed il numero di *pCPU*, i valori di $a^{\%}$ ideali sarebbero: $a_{vm1}^{\%} = 10\%$, $a_{vm2}^{\%} = 20\%$, $a_{vm3}^{\%} = 30\%$, $a_{vm4}^{\%} = 45\%$. La somma di tutti i valori $a^{\%}$ ideali è pari al 105%. L'host è, quindi, in stato di over-commit.
Per questo motivo la quantità di *tu* per ogni macchina virtuale viene riassegnata secondo il metodo sopra esposto.

La Figura 1.5(a) mostra la prima iterazione dell'algoritmo che assegna ad ogni macchina virtuale un massimo del 25% delle risorse. Risulta quindi che $a_{vm1}^{\%} = 10\%$, $a_{vm2}^{\%} = 20\%$, $a_{vm3}^{\%} = 25\%$, $a_{vm4}^{\%} = 25\%$. Mentre *vm1* e

vm2 hanno già soddisfatto la loro *demand*, *vm3* e *vm4* si contendono la capacità di calcolo rimasta, pari al 20%.
Alla seconda iterazione (Figura 1.5(b)) il 20% non ancora allocato viene suddiviso tra le macchine virtuali che necessitano ancora di capacità di calcolo (*vm3* e *vm4*). Viene quindi assegnata loro una quantità di capacità residua massima del 10% e al termine dell'iterazione si avrà: $a^{\%}_{vm1} = 10\%$, $a^{\%}_{vm2} = 20\%$, $a^{\%}_{vm3} = 30\%$, $a^{\%}_{vm4} = 35\%$. L'ultima iterazione (Figura 1.5(c)) è necessaria per poter assegnare a *vm4*, unica macchina a non aver ricevuto una capacità pari alla sua *demand*, il restante 5% di $A^{\%}$.

In questo scenario viene garantita la capacità di calcolo richiesta a tutte le macchina virtuali che fanno una richiesta di $a^{\%} \leq 25\%$. *vm3* riesce ad ottenere ugualmente capacità sufficiente per soddisfare la sua *demand* poiché sfrutta quella non utilizzata da *vm1* e *vm2*. È, quindi, immediato notare che *vm4* vede la sua richiesta di capacità tagliata del 5%, che è il livello di over-commit del sistema.

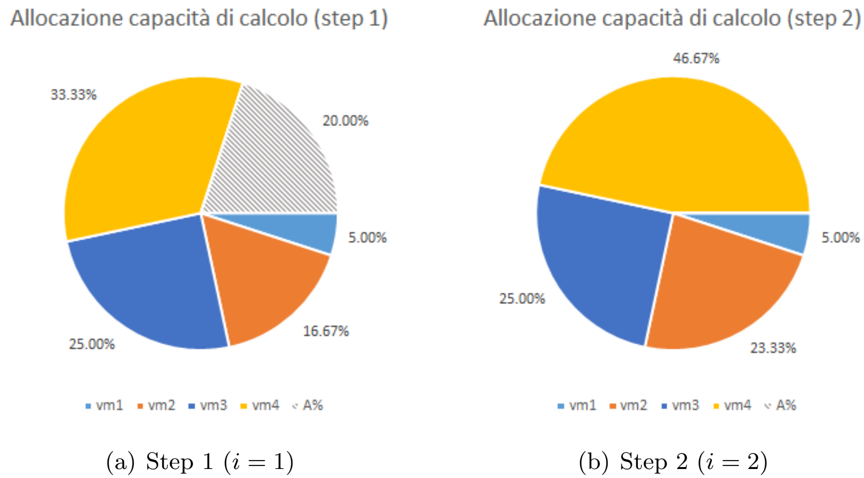
In uno scenario con macchine virtuali con uguale numero di $vCPU$, come quello appena descritto, assegnare lo stesso valore di *Shares* significa garantire a tutte le macchine la medesima capacità di calcolo. Lo stesso criterio può essere applicato anche nel caso in cui esistano macchine virtuali con numero di $vCPU$ differente. Ciò comporta che, in caso di over-commit, l'host fornisca alle macchine virtuali la stessa potenza di calcolo, non considerando il numero effettivo di $vCPU$. Questo fà sì, per esempio, che una macchina single-core utilizzi il processore fisico tanto quanto una macchina dual-core.

Per ovviare a questo inconveniente e per bilanciare meglio la distribuzione della capacità di calcolo, VMware ha scelto di utilizzare valori di *Shares* di default proporzionali al numero di $vCPU$. Uno scenario standard di questa configurazione è quello illustrato nella Figura 1.6 che rappresenta l'applicazione dell'algoritmo sopra esposto in due iterazioni successive per l'allocazione delle risorse di un host con 4 $pCPU$.

Le macchine virtuali *vm1*, *vm2*, *vm3*, *vm4* hanno un numero di $vCPU$ pari a $N^{vCPU}_{vm1} = 2$, $N^{vCPU}_{vm2} = 2$, $N^{vCPU}_{vm3} = 4$ e $N^{vCPU}_{vm4} = 4$ e una *demand* rispettivamente pari a $d^{\%}_{vm1} = 10\%$, $d^{\%}_{vm2} = 50\%$, $d^{\%}_{vm3} = 25\%$, $d^{\%}_{vm4} = 50\%$. Secondo i valori di default di VMware le quattro macchine virtuali hanno *Shares* pari a $S_{vm1} = 2000$, $S_{vm2} = 2000$, $S_{vm3} = 4000$ e $S_{vm4} = 4000$.

Seguendo un ragionamento analogo a quello della Figura 1.5 si possono calcolare i valori di allocazione del sistema che risultano essere: $a^{\%}_{vm1} = 5\%$, $a^{\%}_{vm2} = 23.33\%$, $a^{\%}_{vm3} = 25.00\%$, $a^{\%}_{vm4} = 46.67\%$.

Come ci si aspettava in questo scenario viene penalizzata sicuramente la macchina *vm4*, che ha una richiesta di risorse alta, ma anche *vm2*, che avendo



(a) Step 1 ($i = 1$) (b) Step 2 ($i = 2$)

Figura 1.6: Esempio di suddivisione della capacità di calcolo con macchine virtuali di *Shares* diversi

meno *vCPU*, ha un valore di *Shares* più basso. Da un'altra prospettiva si può notare che questa scelta garantisce la stessa capacità di calcolo ad ogni *vCPU* allocata nell'host.

VMWare ESXi dà la possibilità di cambiare manualmente il livello di *Shares*. In questo lavoro di tesi si è studiato il comportamento atteso delle macchine virtuali variando i livelli. Ciò ha permesso di interpretare al meglio i risultati ottenuti tramite stress-test sulla CPU fisica e di elaborare un modello che valutasse il carico complessivo dell'host. Per le teorie qui sviluppate e gli esperimenti eseguiti, si è scelto di mantenere i livelli di default.

1.4.2 Indicatori di performance

Questa sezione ha lo scopo di elencare e descrivere gli indicatori di performance, sulle *vCPU* e sull'utilizzo dell'host, più rilevanti e più adatti all'obiettivo della tesi. Per ogni voce verrà citata la definizione originale della documentazione di VMware che sarà affiancata da una spiegazione più dettagliata.

Tempo di *Ready* [ms] *Percentage of time that the virtual machine was ready, but could not get scheduled to run on the physical CPU.*
Somma dei tempi in cui le *vCPU* di una macchina virtuale sono stati in *Ready*.

Tempo di *Co-stop* [ms] *Time the virtual machine is ready to run, but is unable to run due to co-scheduling constraints.*

Somma dei tempi in cui le *vCPU* di una macchina virtuale sono stati in *Co-stop*. Per le macchine virtuali single-core questo indicatore è sempre pari a 0.

Tempo di *Wait* [ms] *Total CPU time spent in wait state.*

Somma dei tempi in cui le *vCPU* di una macchina virtuale sono stati in *Wait*.

Tempo di *Run* [ms] *Time the virtual machine is scheduled to run.*

Somma dei tempi in cui le *vCPU* di una macchina virtuale sono stati in *Run*.

Utilizzo dell'host [%] *CPU utilization as a percentage during the interval.*

È la percentuale su il tempo di sample in cui la CPU fisica viene effettivamente utilizzata. Con buona approssimazione si può considerare la somma dei tempi di *Run* delle macchine virtuali presenti nel sistema pari al tempo di utilizzo. Questo perché l'overhead dovuto all'ambiente virtuale, essendo sufficientemente *piccolo*, può essere trascurato.

Uso dell'host [MHz o %] *CPU usage in MHz [or in percentage] during the interval.*

È l'uso in MHz delle *vCPU* presenti nell'host.

Il limite nominale della capacità di calcolo dell'host, ν_{max} , con cui viene calcolato il valore in percentuale, dipende dall'architettura della CPU, in particolare dall'attivazione o meno della tecnologia di hyper-threading.

Dato N_{core}, il numero di core della CPU fisica, e ν, la frequenza nominale, risulta che:

- con HT attivo $\nu_{max} = N_{core} \cdot \nu \cdot K$;
- con HT non attivo $\nu_{max} = N_{core} \cdot \nu$.

Il parametro K, compreso tra 1.2 e 1.3, è stato stimato sulla base di alcuni studi di Intel[7] che dimostrano come l'HT porti un miglioramento delle prestazioni fino al 30%.

Uso della macchina virtuale [MHz o %] *CPU usage in MHz [or in percentage] during the interval.*

Rappresenta la frequenza allocata per la macchina virtuale. La percentuale viene calcolata in rapporto a ν_{vir}, frequenza virtualizzata,

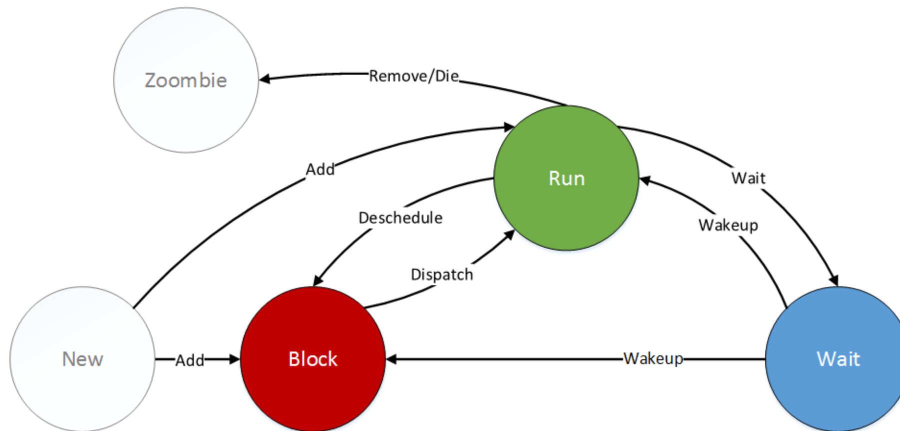


Figura 1.7: Modello a stati generalizzato

data da $\nu_{vir} = N_{vCPU} \cdot \nu$, con N_{vCPU} il numero delle *vCPU* della macchina virtuale.

demand **[MHz]** *The amount of CPU resources a virtual machine would use if there were no CPU contention or CPU limit*
È la quota in MHz richiesta dalla macchina virtuale. VMware non riporta il metodo con cui stima questo valore. Per questo motivo e poiché i dati ottenuti da questo indice risultavano poco affidabili nei momenti in cui l'host si avvicinava allo stato di over-commit, è stato necessario effettuare alcuni test per poter valutare la *demand* delle macchine virtuali in relazione ai tempi *Ready*, *Co-stop*, *Run* e *Wait*.

1.5 Generalizzazione del modello a stati

Il modello proposto da VMware spiega in maniera completa il funzionamento all'interno dello scheduler, ma per le finalità del lavoro di tesi è stato opportuno operare una generalizzazione sulla categorizzazione degli stati.

Di fatto lo stato di *Co-stop* si traduce in un tempo di attesa che impedisce alla *vCPU* di andare in esecuzione. Fermo restando che le finalità dello *stop* impartito alla *vCPU* sono sicuramente diverse da quelle che portano lo scheduler a mettere una *vCPU* in stato di *Ready*, si è deciso di considerare entrambi gli stati come uno solo definito stato di *Block*.

La macchina a stati cui farà riferimento questo elaborato è mostrata in Figura 1.7. Questa semplificazione porta dei vantaggi notevoli considerando che in un qualsiasi ambiente di calcolo virtuale gli stati, seppur diversi a

seconda dall'architettura, devono poter essere suddivisi fondamentalmente in tre categorie:

Run la *vCPU* sta utilizzando la CPU fisica;

Block la *vCPU* vorrebbe utilizzare la CPU fisica, ma non è al momento disponibile disponibili;

Wait la *vCPU* è in attesa delle risorse di I/O o è in stato di Idle.

Nei capitoli successivi la somma del tempo di *Ready* e del tempo di *Co-stop*, verrà indicata solo con il termine tempo di blocco.

Definito lo schema generalizzato, è possibile impostare uno studio più approfondito per comprendere e affinare ulteriormente il modello che descrive lo scheduler in stato di over-commit. Infatti, mentre il tempo di *Run* di ciascuna macchina virtuale è stimabile con ottime approssimazioni poiché è noto l'algoritmo di scheduling (Sezione 1.4.1), si ha più incertezza nella previsione del tempo di *Block*. Lo studio di questo parametro risulta importante in quanto influisce direttamente sulle performance della macchina virtuale.

Il capitolo seguente, sfruttando il modello a stati generalizzato, descrive i test effettuati per ottenere una descrizione affidabile del comportamento dello scheduler in stato di over-commit.

Capitolo 2

Test e prime osservazioni

Questo capitolo descrive gli obiettivi dei singoli test, la metodologia con cui ognuno di essi è stato realizzato e le osservazioni utili alla creazione di un modello capace di rappresentare la relazione tra il tempo di blocco, il tempo di run e il tempo di attesa. In questo modo si cercherà di fornire una stima della *demand* delle macchine virtuali.

Come già anticipato nella Sezione 1.4.2 del capitolo 1, il valore della *demand* non è misurabile utilizzando soltanto i dati che si possono ottenere monitorando la macchina virtuale come un sistema black-box.
Per ottenere una stima affidabile si è deciso di procedere in tre fasi successive:

- realizzazione di test in cui, nota la *demand*, si misuravano il tempo di blocco e il tempo di attesa
- proposta di una relazione tra i valori
- test di verifica della proposta

La prima parte del capitolo introduce gli strumenti utilizzati e lo stato dell'ambiente a *riposo*. In seguito verranno prese in esame due categorie di test che hanno permesso di profilare la crescita del tempo di blocco, al variare del numero di $vCPU$, o più in generale del numero di macchine virtuali, e della *demand*.

Considerando il sistema come un modello chiuso, in cui il numero di macchine virtuali non vari durante l'arco del test, era lecito aspettarsi che la crescita del tempo di blocco fosse inquadrabile essenzialmente in due fasi:

fase di regime normale - fase in cui la *demand* è *sufficientemente* bassa da non richiedere l'intervento del sistema a forzare l'attesa e il tempo di utilizzo della CPU;

fase di saturazione - fase in cui il sistema forza l'attesa, quindi il tempo di blocco si stabilizza prima di rendere disponibile la risorsa per un tempo anch'esso forzato.

Riconoscere le due fasi ha una notevole importanza poiché, nel momento in cui è necessario stimare la *demand* richiesta dalla macchina virtuale, la fase di regime normale individua un intervallo nel quale il valore della *demand* stessa può essere approssimata, con un errore accettabile, alla percentuale del tempo di run sul tempo di sample.

La fase di saturazione, invece, individua l'intervallo in cui il tempo di run è imposto dal sistema in base al numero di *vCPU* che richiedono potenza di calcolo. In questa fase la *demand* è impredicibile infatti anche i log di VMware riportano valori che variano in maniera inaspettata, nonostante la *demand* rimanga costante. Gli esperimenti mirano ad individuare gli estremi entro cui sia possibile collocare la *demand* della macchina virtuale conoscendo tempo di run, tempo di blocco e tempo di attesa.

Per lo studio delle due fasi i test sono stati impostati, inizialmente, per capire quale fosse il livello di saturazione del tempo di blocco, quindi si è passati allo studio della fase di regime normale. Nella descrizione degli stessi, infatti, si comincerà ad analizzare i risultati ottenuti con una *demand* del 100% al variare del numero di macchine virtuali con 1, 2, poi con 3 e quindi con 4 *vCPU*.

Questi ultimi sono stati più approfonditi, così, da comprendere al meglio anche la fase di regime normale prima, cioè, che il tempo di blocco saturasse.

2.1 Strumenti e ambiente di test

Per la realizzazione dei test HP ha messo a disposizione l'intero ambiente VMWare vSphere nella versione 5 accessibile tramite vSphere Client. In particolare l'host che ha ospitato le macchine virtuali è un HP ProLiant DL360 G7 con processore Intel® Xeon® E5620 su cui c'è installato VMWare ESXi 5. Da notare che durante tutti i test la funzionalità di Hyper-Threading (HT) di cui è dotata la CPU Intel è rimasta attiva, pertanto, data l'architettura quad-core, l'ambiente contava 8 *pCPU*. La Tabella 2.1 riassume tutti i dati della configurazione hardware.

Per ciò che riguarda le macchine virtuali, è stata creata un'immagine di Red Hat Enterprise Linux 6 Server con installati, oltre ai pacchetti di base:

`stress`**[12]** - programma che permette di generare carico di lavoro (CPU, Memoria, I/O, Disco) su sistemi POSIX;

Configurazione host	
Produttore	HP
Modello	ProLiant DL360 G7
Processori	4 CPU x 2.4 GHz
Tipo di processore	Intel(R) Xeon(R) CPU E5620 @ 2.40GHz
Hyper-Threading	Attivo
Tot Memoria	35.99 GB

Tabella 2.1: Configurazione hardware dell'host

`cpulimit`[4] - programma che limita ad un processo specificato l'utilizzo della CPU ad un valore espresso in percentuale.

L'utilizzo di un script di bash, noto come `CPU load script`[1], che sfrutta i due pacchetti sopra descritti, ha permesso di *stressare* in maniera uniforme le $vCPU$ delle singole macchine virtuali.

Ogni test è stato realizzato con macchine virtuali con lo stesso numero di $vCPU$. Partendo dai casi con 1 $vCPU$ si è passati a macchine virtuali con 2, 3 e infine 4 $vCPU$.

Per via di alcune esigenze di HP, durante i test, sono dovute rimanere attive sei macchine virtuali. Nonostante ciò si è scelto di utilizzare lo stesso l'ambiente di test in quanto si è potuto lavorare tenendo conto di un carico di lavoro reale preso da un contesto industriale.
Questa scelta ha reso necessario uno studio preliminare del comportamento preliminare delle macchine virtuali esistenti.

Nella sezione seguente verrà valutato in maniera qualitativa l'impatto di esse sui test effettuati così da avere un'ottica completa per valutare, poi, i risultati.

2.1.1 Macchine virtuali esistenti

Le macchine virtuali attive nell'host sono sei in totale: cinque di queste hanno 2 $vCPU$, la restante 1 $vCPU$. Dai report reperibili tramite vSphere Client con un tempo di sample di $20s$ risulta che il tempo di run totale ha un valore medio attorno agli $11.05s$.

In modo abbastanza immediato, noto t^{sample}, un certo tempo di sample, N^{pCPU}, numero di $pCPU$, è ragionevole stimare t^{run}, tempo di run totale fornito dal sistema nell'intervallo, con la formula

$$t^{run} = N^{pCPU} \cdot t^{sample}$$

Data l'architettura dell'host il tempo d'esecuzione a disposizione dei test è di circa $8 \cdot 20s - 11.05s = 148.95s$ per ogni intervallo di sample che equivale a circa il 93% del tempo d'esecuzione totale. È prevedibile, quindi, che il modello finale sottostimi il tempo di blocco misurato sulle macchine virtuali di test, ma che questa differenza sia un numero *sufficientemente piccolo* a validare il modello. Le evidenze sperimentali descritte nelle Sezioni 2.2.4 e 2.3.1 hanno poi confermato questa ipotesi.

2.2 Test per il livello di saturazione

Dovendo capire quale fosse il valore di saturazione per il tempo di blocco era ragionevole aspettarsi risultati di una certa rilevanza solo nelle configurazioni che prevedevano un numero totale di *vCPU* superiore ad 8, numero di *pCPU* offerto dal sistema. Si è quindi deciso, una volta scelta l'architettura virtuale da studiare, di valutare il tempo di blocco incrementando il numero di macchine virtuali, fino ad un massimo di sei. Il limite di sei macchine è dovuto alle dimensioni del datastore dell'ambiente VMware di test.

Qui di seguito verranno illustrati i risultati dei test effettuati mantenendo la *demand* di ogni *vCPU* delle macchine virtuali di test al 100%.

Per una facile lettura dei dati, i grafici riportano sull'asse delle ascisse il numero di macchine virtuali che richiedono l'uso della CPU e sull'asse delle ordinate il tempo di blocco espresso in millisecondi per la singola *vCPU*. Il tempo di sample con cui sono stati presi i dati è di $20s$. Sul lato destro del grafico è presente anche la scala in percentuale del tempo di blocco rispetto al tempo di sample.

2.2.1 Macchine virtuali con 1 *vCPU*

I test con macchine virtuali single-core hanno effettivamente confermato l'ipotesi precedente dando risultati poco significativi per la realizzazione del modello finale. Il tempo di blocco ha sempre avuto valori molto bassi indipendentemente dal numero di macchine virtuali (da 1 fino a 6).

Un'effettiva crescita del tempo di blocco c'è stata passando dai $7ms$, ottenuti con 1 macchina virtuale, ai $78ms$, ottenuti, invece, con 6 macchine virtuali, ma è ovvio che rapportato al tempo di sample il tempo di blocco è ≈ 0.

Da notare che avendo una sola *vCPU*, era naturale aspettarsi che le macchine virtuali non registrassero alcun ritardo di *Co-stop*. Di fatto in questi test il tempo di blocco ottenuto è dato solamente dal *Ready*.

La Figura 2.1 riporta il grafico che riassume le osservazioni fatte sopra.

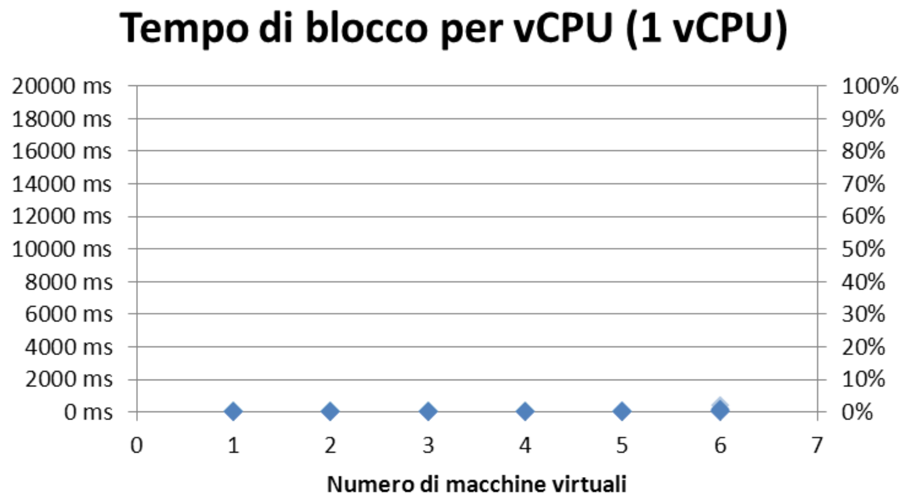


Figura 2.1: Variazione del tempo di blocco al variare del numero di macchine virtuali con 1 *vCPU*

2.2.2 Macchine virtuali con 2 *vCPU*

Per poter ottenere risultati maggiormente significativi i test realizzati con macchine virtuali da 2 *vCPU* sono stati fatti a partire da 2 macchine virtuali attive.

I risultati ottenuti hanno riscontrato un aumento del tempo di blocco, questa volta comprendente il *Co-stop*, già con 3 macchine virtuali, per una richiesta totale di 6 *vCPU* al 100%. Il tempo di blocco misurato sfiora i $2s$ (10% del tempo di sample).

I dati non smentiscono l'ipotesi sopra che affermava che il valore del tempo di blocco fosse trascurabile finché il numero totale di *vCPU* fosse minore o uguale al numero di *pCPU* del sistema, in questo caso fino a 4 macchine virtuali.

Il primo dato rilevante è il tempo di blocco misurato con 5 macchine virtuali che, come illustrato dal grafico in Figura 2.2, si attesta attorno al 20% per un totale di circa $4s$.

Viene introdotta da qui in avanti un notazione che aiuterà a mettere in relazione una grandezza misurata x con una grandezza stimata $\hat{x}$. Più formalmente, la stima di un certo valore x viene fornita a meno di un certo errore ϵ_x.
Si ha quindi:

$$x = \hat{x} + \epsilon_x \tag{2.1}$$

Alla luce di ciò è possibile analizzare i dati ottenuti con 5 macchine virtuali. Il risultato ottenuto, infatti, trova la seguente giustificazione formale.

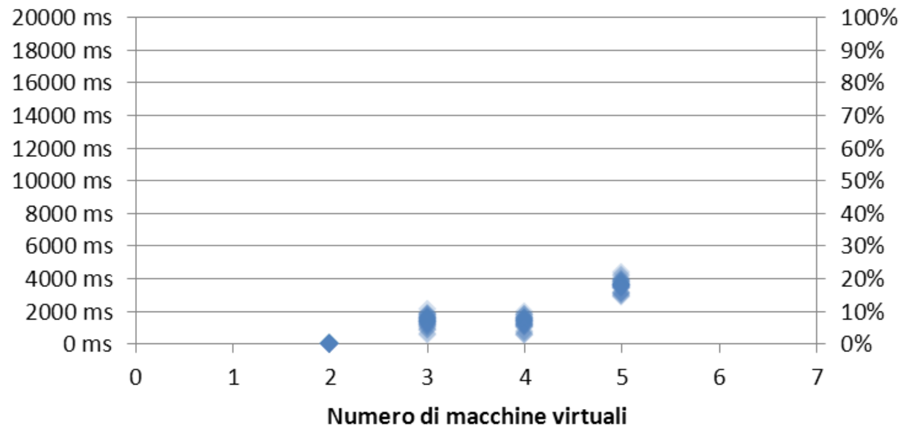


Figura 2.2: Variazione del tempo di blocco al variare del numero di macchine virtuali con 2 *vCPU*

Considerando la virtualizzazione di 10 *vCPU* con una *demand* del 100% ed avendo il sistema solo 8 *pCPU*, è possibile stimare $\hat{R}^{\%}$ [1], tempo di run in percentuale per singola *vCPU*, con la formula

$$\hat{R}^{\%} = \frac{N^{pCPU}}{N^{vCPU}} \tag{2.2}$$

In questa situazione, poiché le macchine virtuali fanno una richiesta continua di utilizzo, è anche ragionevole stimare il tempo di attesa pari a 0:

$$\hat{W}^{\%} = 0 \tag{2.3}$$

Noto che, per definizione,

$$\hat{R}^{\%} + \hat{B}^{\%} + \hat{W}^{\%} = 1 \tag{2.4}$$

si ha che, data la 2.2 e la 2.3, il tempo di blocco in fase di saturazione si può stimare con la formula seguente

$$\hat{B}^{\%} = 1 - \hat{R}^{\%} = 1 - \frac{N^{pCPU}}{N^{vCPU}} \tag{2.5}$$

In conclusione per il test in esame con 5 macchine virtuali ci si aspettava un rapporto tra il tempo di blocco e il tempo di sample pari a $1 - \frac{8}{10} = 20\%$.

[1] Da qui in avanti le lettere maiuscole indicheranno valori in fase di saturazione.

2.2.3 Macchine virtuali con 3 *vCPU*

I test effettuati con macchine virtuali da 3 *vCPU* hanno cominciato a dare alcuni risultati più interessanti. In particolare per 1 e 2 macchine virtuali il tempo di blocco rimane ≈ 0, per 3 e 4, invece, assume valori, rispettivamente, attorno al 18% e al 40%. Secondo la teoria i valori attesi del tempo di blocco per le due configurazioni sono pari all' 11% (3 macchine virtuali) e al 30% (4 macchine virtuali).

Questa differenza può essere giustificata dal consumo dal tempo di run da parte delle macchine già esistenti sull'host. Infatti considerando che il tempo di run dell'host residuo, calcolato nella Sezione 2.1.1, vale $t_{res}^{run} \approx 149s$, e che, essendo la *demand* pari al 100%, ad ogni *vCPU* viene assegnato un tempo di run $\hat{R} = \frac{149s}{N^{vCPU}} = \frac{149s}{9} \approx 16.6s$. Da cui si può calcolare $\hat{R}^{\%} = \frac{16.6s}{20s} \approx 83\%$.
Ricordando la 2.3, la 2.5 e la 2.4, si ha che $\hat{B}^{\%} = 1 - \hat{R}^{\%} \approx 17\%$.

Con un ragionamento analogo si può calcolare che nel caso di 4 macchine virtuali $\hat{B}^{\%} \approx 38\%$.

È evidente che in questa configurazione l'influenza delle macchine virtuali esistenti non è ancora trascurabile.

Nonostante ciò i risultati dell'ultimo test hanno messo in luce una prima relazione tra tempo di blocco e tempo di run che cercherà, poi, una conferma definitiva negli esperimenti con macchine virtuali quad-core.

Osservazione Dato N^{vCPU} il numero di *vCPU*, tutti con la stessa priorità, con una *demand* pari al 100%, dato N^{pCPU} il numero di *pCPU* del sistema, con $N^{vCPU} > N^{pCPU}$, si può definire $\hat{B}^{\%}$, la percentuale di tempo di blocco, e $\hat{R}^{\%}$, la percentuale del tempo di run, per ogni *vCPU* come

$$\hat{B}^{\%} = 1 - \frac{N^{pCPU}}{N^{vCPU}}$$

$$\hat{R}^{\%} = \frac{N^{pCPU}}{N^{vCPU}}$$

2.2.4 Macchine virtuali con 4 *vCPU*

L'intuizione avuta con i test con macchine virtuali triple-core ha dettato le linee guida per gli esperimenti su quelle quad-core. Partendo da configurazioni che permettessero di avere un numero di *vCPU* maggiore del numero

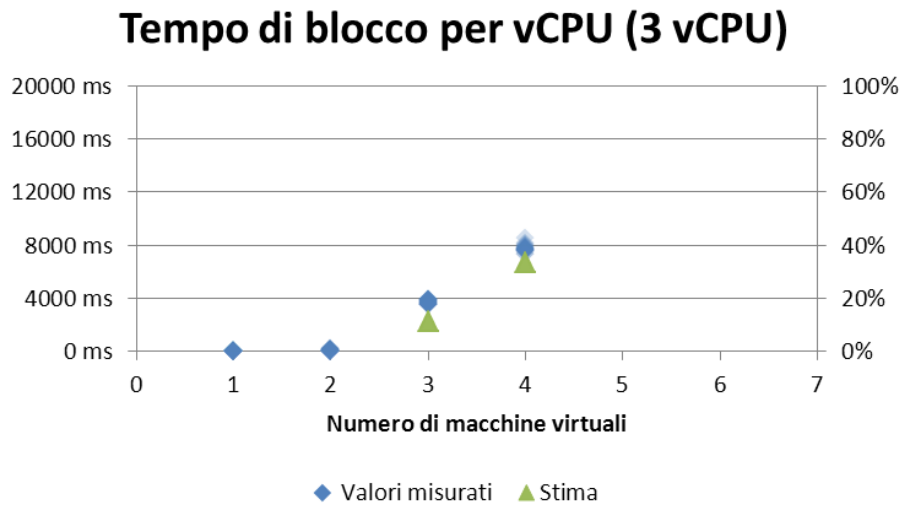


Figura 2.3: Variazione del tempo di blocco al variare del numero di macchine virtuali con 3 *vCPU*

di *pCPU*, si è riusciti a forzare il sistema in stato di *overcommit* già con l'utilizzo di 3 macchine virtuali.

La Tabella 2.2 mette a confronto la stima e la media dei dati misurati del tempo di blocco in valore percentuale, evidenziando un'effettiva corrispondenza che conferma l'ipotesi avanzata.

Si può osservare che, nonostante la presenza delle macchine virtuali disturbino i risultati dei test, essendo la crescita della stima del tempo di blocco asintotica al 100% (per una numero infinito di *vCPU* il tempo di blocco coincide con il tempo di sample), la differenza tra stima e dati diminuisce all'aumentare del numero di *vCPU*.

# VM	# *vCPU*	$\hat{B}$	B
3	12	33%	38%
4	16	50%	54%
5	20	60%	63%
6	24	67%	69%

Tabella 2.2: Confronto stima e misura del tempo di blocco medio

Per una lettura qualitativa dei dati è stato riportato in Figura 2.4 il grafico della variazione del tempo di blocco, stimato e misurato, con l'aumento del numero di macchine virtuali.

Oltre al funzionamento del sistema in fase di saturazione, per la creazione del modello era importante studiare la crescita del tempo di blocco in

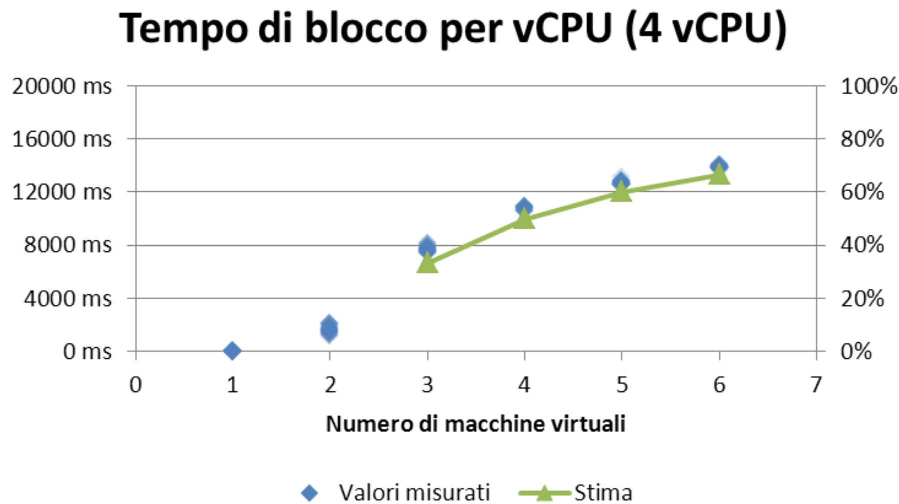


Figura 2.4: Variazione del tempo di blocco al variare del numero di macchine virtuali con 4 *vCPU*

relazione alla crescita della *demand*. La sezione seguente descrive gli esperimenti fatti a questo scopo e propone un'interpretazione dei risultati. In questo modo sarà possibile completare la modellazione del sistema e fornire una relazione che lega la *demand*, il tempo di blocco e il tempo di run.

2.3 Test per la fase di regime normale

I primi test finalizzati alla fase di regime normale hanno visto lo studio del comportamento di macchine virtuali quad-core realizzato con questa modalità: sfruttando lo script di bash, che permette di inserire la durata del tempo di stress, si è fatto sì che nell'arco di un'ora ogni 15 minuti una macchina virtuale terminasse di richiedere potenza di calcolo. Facendo partire simultaneamente 4 macchine virtuali con una *demand* omogenea si è riusciti a confrontare i dati in 4 fasi successive.

Un'idea del funzionamento del test viene data dalla Figura 2.5 che mostra il tempo di run delle 4 macchine virtuali di test con una *demand* del 100%.

Il grafico mostra come, dopo la fine della richiesta della macchina virtuale numero 1 (t = 0:12), il sistema di scheduling ridistribuisca la capacità di calcolo tra le altre 3 macchine rimaste attive.

Successivamente (t = 0:41), invece, quando l'unica macchina virtuale a rimanere attiva è la numero 4, si nota come non vi sia più un limite dovuto allo stato di overcommit. La stabilizzazione del tempo di run è sintomo di un comportamento non più legato a costrizioni provenienti dall'esterno.

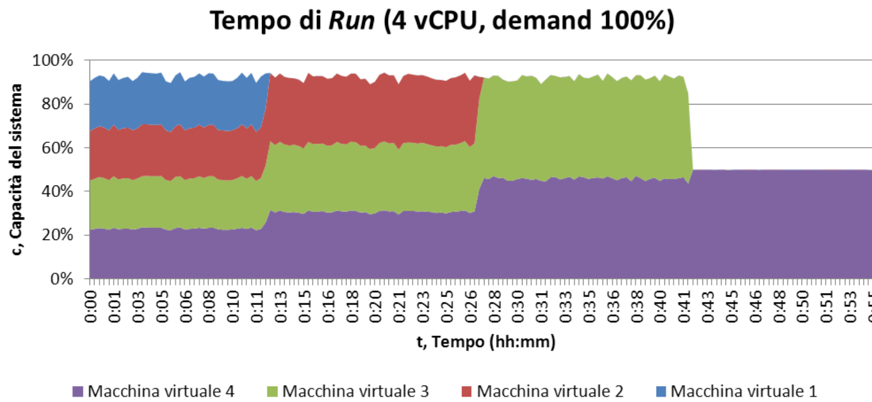


Figura 2.5: Tempo di *Run* durante il test di 4 macchine virtuali con *demand* pari al 100%

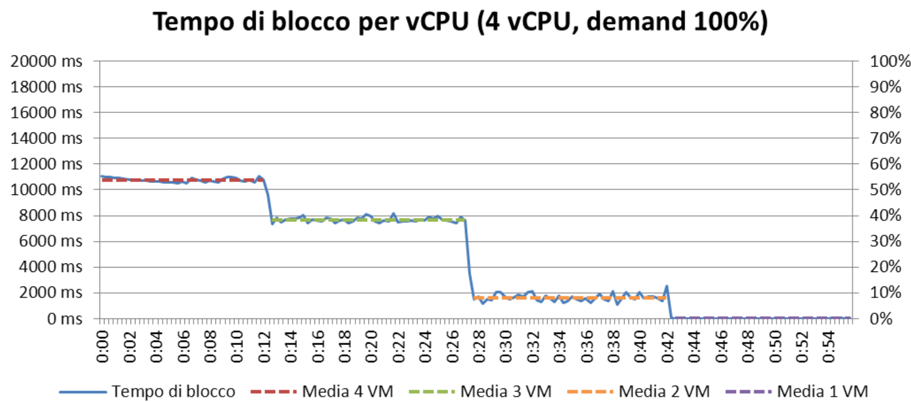


Figura 2.6: Tempo di blocco durante il test di 4 macchine virtuali con *demand* pari al 100%

Il fatto che il sistema non sia più in overcommit si può dedurre anche dal valore del tempo di blocco (vedi grafico in Figura 2.6) che, con 1 macchina virtuale attiva, assume un valore $\approx 0\%$.

Ad un'osservazione più attenta si può notare che nella fase in cui vi sono solo 2 macchine virtuali il tempo di blocco assume un valore non trascurabile vicino all'8%. Questo valore è imputabile alle macchine virtuali esistenti nell'ambiente.

Infatti, considerando t_{res}^{run}, il tempo di run residuo già calcolato nella Sezione 2.1.1, e tenendo conto che le macchine virtuali di test per soddisfare la loro *demand* necessitano di $20s$ d'esecuzione per ogni $vCPU$, si può considerare che il tempo di run dedicato ad ogni $vCPU$ sia pari a

$$\hat{r} = \frac{t_{res}^{run}}{8} \approx 18.75s$$

Da cui si può calcolare che

$$\hat{r}^{\%} = \frac{18.75}{20} \approx 0.94$$

Noto che $\hat{w}^{\%} = 0$, la percentuale stimata del tempo di blocco risulta pari

$$\hat{b}^{\%} = 1 - \hat{r}^{\%} \approx 0.06$$

Il valore trovato si avvicina alle misure rilevate. Una volta chiara la complicanza derivata dalle macchine esistenti si è deciso di effettuare i test evitando casi limite e concentrandosi su configurazioni con un numero di macchine virtuali quad-core superiore a due.

In un primo momento le misurazioni del tempo di blocco sono state effettuate facendo una richiesta di risorse pari al 25%, 50%, 75% e 100%. Essendo tutte e 4 le macchine dotate di 4 *vCPU* è stato immediato riconoscere il punto di overcommit ad un valore di *demand* superiore al 50%.

I test successivi si sono concentrati su valori della *demand* nell'intorno del 50%: 37%, 43%, 53%, 56%, 62%.

Nelle sezioni successive verranno illustrate prima le misurazioni ottenute con 4 macchine virtuali, fornendo una graduale interpretazione dei risultati. Successivamente le ipotesi avanzate nel caso di 4 macchine virtuali verranno valutate nei casi di 3, 5 e 6 macchine virtuali.

A conclusione della sezione si avranno gli strumenti necessari alla creazione di un modello capace di stimare il tempo di blocco in fase di regime normale.

2.3.1 Fase di regime normale con 4 macchine virtuali

Gli esperimenti fatti con 4 macchine virtuali con una *demand* crescente hanno evidenziato la fine della fase di regime normale e l'inizio di quella di saturazione per valori superiori $\approx 56\%$. Come si nota dalla Tabella 2.3 per questo intervallo il tempo di blocco si stabilizza nell'intorno del 50% del tempo di sample.

Una prima interpretazione dei dati viene data considerando l'architettura hardware dell'host: avendo 8 *pCPU*, il sistema è in grado di eseguire contemporaneamente solo 2 macchine virtuali quad-core. In maniera ancora più semplicistica si può pensare che 2 macchine virtuali utilizzino sempre le stesse 4 *pCPU*.

Lo scenario sopra descritto può essere riassunto con i concetti di *stazione* e *cliente*: le stazioni sono i "posti" di cui è dotato il sistema, il cliente è colui che occupa una stazione.

Demand	$b_{\%}$	Tempo di blocco
0%	0%	2 ms
25%	1%	213 ms
37%	8%	1580 ms
43%	16%	3163 ms
50%	34%	6798 ms
53%	49%	9817 ms
56%	54%	10822 ms
62%	54%	10801 ms
75%	54%	10760 ms
100%	54%	10766 ms

Tabella 2.3: Crescita tempo di blocco con 4 macchine virtuali

4 *pCPU* per 2 macchine virtuali con 4 *vCPU* ciascuno si traduce in un modello ad 1 stazione e 2 clienti.

La *demand* delle macchine virtuali diventa la probabilità che una di esse richieda una stazione. Partendo da questi concetti è possibile impostare alcuni modelli che aiuteranno a definire il comportamento del tempo di blocco rispetto alla *demand*.

Di seguito i modelli stazione-cliente verranno affiancati dalla notazione N_s/N_c dove N_s è il numero di stazioni e N_c il numero di clienti.

Modello stazione-cliente 1/2

In prima istanza si è deciso di tracciare una stima del tempo di blocco semplicemente sfruttando la formula seguente:

$$\hat{b}^{\%} = {d^{\%}}^{2}$$

dove

$d^{\%}$ è il valore della *demand* delle macchine virtuali, espresso in percentuale

$\hat{b}^{\%}$ è la percentuale di tempo di blocco nell'intervallo di tempo considerato

La percentuale di tempo di blocco diventa semplicemente la probabilità che entrambe le macchine virtuali vogliano utilizzare l'unica stazione disponibile. Superato il valore limite del 50%, il sistema interviene in modo da alternare l'utilizzo della stazione tra le due macchine virtuali.

Il modello ottenuto, però, presenta diversi difetti:

- il valore di saturazione, vicino al 75%, è ben lontano da quello misurato

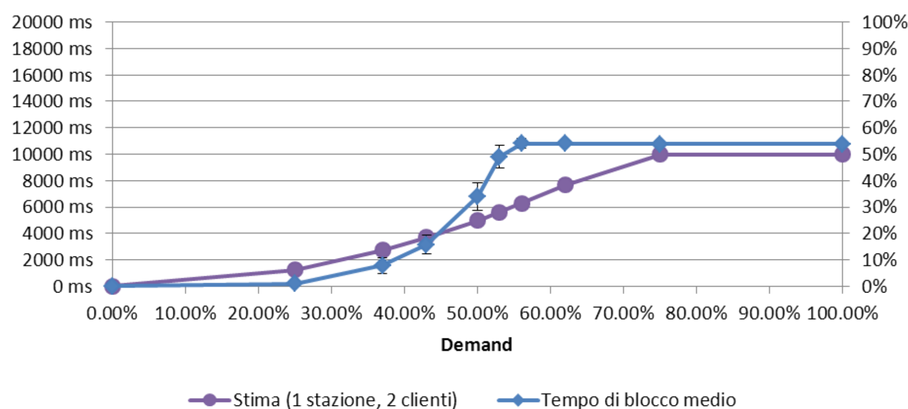


Figura 2.7: Confronto tempo di blocco misurato e stima 1/2

- per i valori di *demand* inferiori al 43%, il tempo di blocco è sovrastimato
- per i valori di *demand* superiori al 43%, il tempo di blocco è sottostimato

Le osservazioni sopra riportate sono evidenziate anche dal grafico in Figura 2.7. Per questo si è stati portati a considerare un modello lievemente più complesso.

Modello stazione-cliente 2/4

Considerando la tipologia quad-core delle macchine virtuali presenti nell'host e considerando il numero totale di $pCPU$, il numero di stazioni che può fornire il sistema è pari a 2. Avendo 4 macchine virtuali attive è possibile considerare un modello stazione-cliente 2/4.

La stima del tempo di blocco si basa questa volta sulla probabilità di trovare occupate entrambe le stazioni del modello. Avendo 4 clienti la formula per il calcolo del $\hat{b}^{\%}$ è:

$$\hat{b}^{\%} = 4 \cdot {d^{\%}}^{3} \cdot \left(1 - d^{\%}\right) + {d^{\%}}^{4}$$

Il valore di $\hat{b}^{\%}$ questa volta è dato dalla probabilità che 3 macchine virtuali su 4 richiedano l'uso di una stazione sommata alla probabilità che a richiedere una stazione siano tutte e 4 le macchine. Il coefficiente moltiplicativo si giustifica perché nel caso di 3 macchine virtuali, è possibile trovare una qualsiasi delle 4 macchine virtuali a riposo.

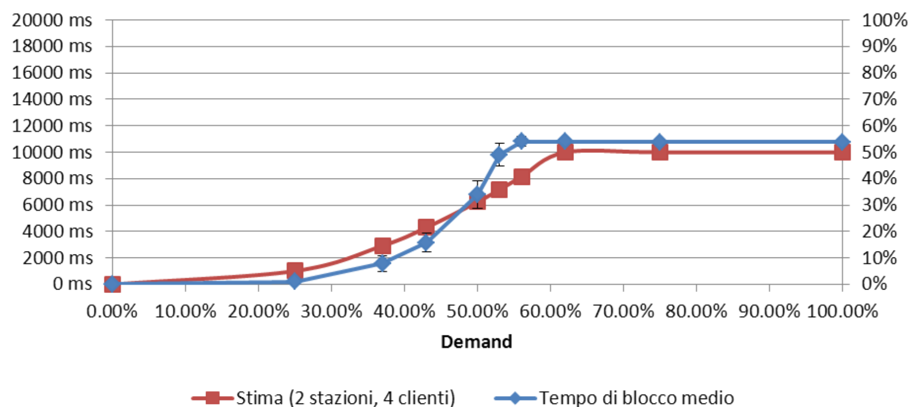


Figura 2.8: Confronto tempo di blocco misurato e stima 2/4

Così facendo si ottiene un andamento del tempo di blocco più vicino a quello misurato, ma, come mostra il grafico in Figura 2.8, non ancora soddisfacente.

L'ipotesi su cui si fondano i modelli stazione-cliente 1/2 e 2/4 deriva dal fatto che una macchina virtuale, per poter ottenere potenza di calcolo, debba avere a disposizione tutte le $vCPU$ della sua configurazione. Da quanto visto nella Sezione 1.4 del Capitolo 1, però, questo accadeva nelle versioni VMWare ESXi inferiori alla 3, quando veniva applicato il concetto di Co-scheduling stretto. La versione 5 di VMWare ESXi, utilizzata durante i test, invece, permette l'esecuzione anche di singole $vCPU$, mettendo in stato di *Block* quelle della stessa macchina virtuale.

Questo e lo scarto tra i modelli e i valori misurati ha fatto sì che venisse considerata l'ipotesi di supporre il comportamento di ogni $vCPU$ totalmente indipendente dalla macchina virtuale cui appartiene.

Modello stazione-cliente 8/16

L'ipotesi di considerare ogni $vCPU$ indipendente dalla macchina virtuale cui appartiene ha fatto sì che il modello stazione-cliente dovesse considerare 8 stazioni (8 $pCPU$) per 16 clienti (16 $vCPU$). La complessità del modello ha portato ad una formalizzazione della formula per il calcolo del tempo di blocco che risulta essere:

$$\hat{b}^{\%} = \sum_{i=0}^{\Delta_{xCPU}-1} \binom{N^{vCPU}}{i} \cdot \left(1 - d^{\%}\right)^{i} \cdot d^{\%^{N^{vCPU}-i}} \tag{2.6}$$

dove Δ_{xCPU}, è la differenza tra N^{vCPU}, numero di $vCPU$ totale, e N^{pCPU}, numero di $pCPU$ totale del sistema.

La formula precedente è una generalizzazione di quella mostrata nella sotto sezione precedente e deriva direttamente dalla teoria del calcolo delle probabilità. Più precisamente si suppone che la probabilità di avere k $vCPU$ che non richiedano l'uso di $pCPU$ abbia una densità binomiale $Bi(n, p)$ con parametri $n = N^{vCPU}$ e $p = 1 - d^{\%}$, o semplicemente $X \sim Bi(N^{vCPU}, 1 - d^{\%})$.

Per poter valutare la probabilità di trovare il sistema in stato di overcommit è necessario calcolare $P(X \leq N^{vCPU} - N^{pCPU}) = P(X \leq \Delta_{xCPU})$ da cui la formula

$$P(X \leq \Delta_{xCPU}) = \sum_{i=0}^{\Delta_{xCPU}-1} \binom{N^{vCPU}}{i} \cdot \left(1 - d^{\%}\right)^{i} \cdot {d^{\%}}^{N^{vCPU}-i}$$

Presupposto di questa teoria è la totale indipendenza nella richiesta di utilizzo delle risorse da parte delle singole $vCPU$.

Alla luce di quest'ultima interpretazione i valori misurati del tempo di blocco hanno avuto un'effettiva vicinanza ai valori stimati. La Tabella 2.4 mette a confronto i 3 modelli evidenziando il punto in cui comincia la fase di saturazione.

Il modello 8/16, come mostra anche il grafico in Figura 2.9, garantisce un'aderenza maggiore ai risultati soprattutto nell'intervallo che vede la *demand* passare dal 25% al 53%, cosa che non si riscontra negli altri due modelli.

I risultati con 4 macchine virtuali sembrano confermare l'ipotesi dell'indipendenza delle $vCPU$. Per un'ulteriore conferma si è deciso di confrontare la stima del valore del tempo di blocco con i dati ottenuti dai i test di 3, 5, e 6 macchine virtuali.

2.3.2 Validazione della fase di regime normale e conclusioni

Per vagliare la teoria avanzata si è potuto sfruttare i dati dei test con 3 macchine virtuali attive già ottenuti con gli stessi valori di *demand* impostati per lo studio precedente.

Il confronto tra stima e risultati, come mostra il grafico in Figura 2.10, ha confermato ancora una volta come l'idea di ritenere indipendente ogni $vCPU$ delle macchine virtuali permetta di interpretare al meglio il comportamento dell'ambiente virtuale.

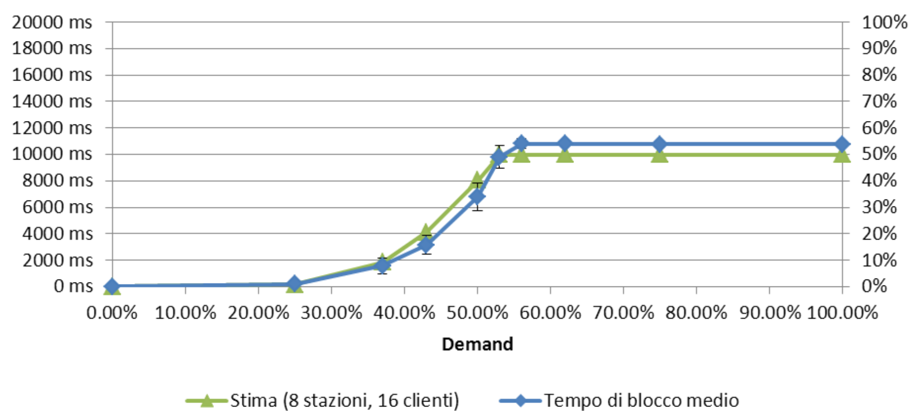


Figura 2.9: Confronto tempo di blocco misurato e stima 8/16

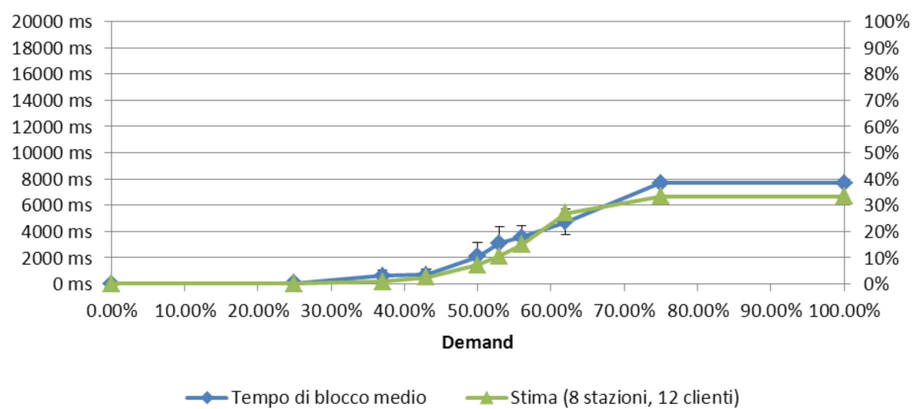


Figura 2.10: Validazione modello stazione-cliente con 3 maccine virtuali

Demand	$b_{\%}$ medio	$\hat{b}^{\%}$ 1-2	$\hat{b}^{\%}$ 2-4	$\hat{b}^{\%}$ 8-16
0.00%	0.01%	0.00%	0.00%	0.00%
25.00%	1.06%	6.25%	5.08%	0.75%
37.00%	7.90%	13.69%	14.64%	9.26%
43.00%	15.81%	18.49%	21.55%	20.60%
50.00%	33.99%	25.00%	31.25%	40.18%
53.00%	49.09%	28.09%	35.88%	49.81%
56.00%	54.11%	31.36%	40.74%	50.00%
62.00%	54.01%	38.44%	50.00%	50.00%
75.00%	53.80%	50.00%	50.00%	50.00%
100.00%	53.83%	50.00%	50.00%	50.00%

Tabella 2.4: Confronto modelli stazione-cliente 1/2, 2/4 e 8/16

Usando la stessa tecnica di test sfruttata per ottenere i dati con 1, 2, 3, 4 macchine virtuali con stessa *demand*, sono stati svolti degli esperimenti con 5 e 6 macchine virtuali. I risultati di quest'ultimi sono mostrati nei grafici in Figura 2.11.

Le tabelle 2.5 e 2.6 riassumono i risultati ottenuti con tutti i test mettendo in evidenza b, il tempo di blocco medio, $\hat{b}$, la stima del tempo di blocco, σ, la deviazione standard del tempo di blocco, e ϵ, l'errore in valore assoluto tra b e $\hat{b}$.

Tutti i test effettuati hanno messo in luce una relazione evidente tra la *demand* ed il tempo di blocco medio nel caso in cui $N^{vCPU} \geq N^{pCPU}$, relazione che può essere riassunta in questo modo:

$$\hat{B}^{\%} = 1 - \frac{N^{pCPU}}{N^{vCPU}} \tag{2.7}$$

$$\hat{b}^{\%} = \begin{cases} \sum_{i=0}^{\Delta_{xCPU}-1} \binom{N^{vCPU}}{i} \cdot \left(1 - d^{\%}\right)^{i} \cdot d^{\%\, N^{vCPU}-i} & \hat{b}^{\%} < \hat{B}_{\%} \\ \hat{B}^{\%} & altrimenti \end{cases} \tag{2.8}$$

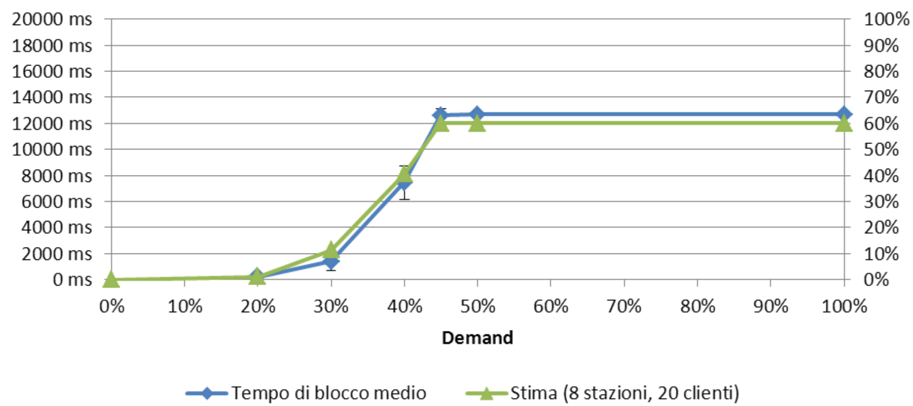
Da cui si possono anche calcolare il tempo di run con la formula:

$$\hat{r}^{\%} = \begin{cases} d^{\%} & d^{\%} < 1 - \hat{B}^{\%} \\ 1 - \hat{B}^{\%} & altrimenti \end{cases} \tag{2.9}$$

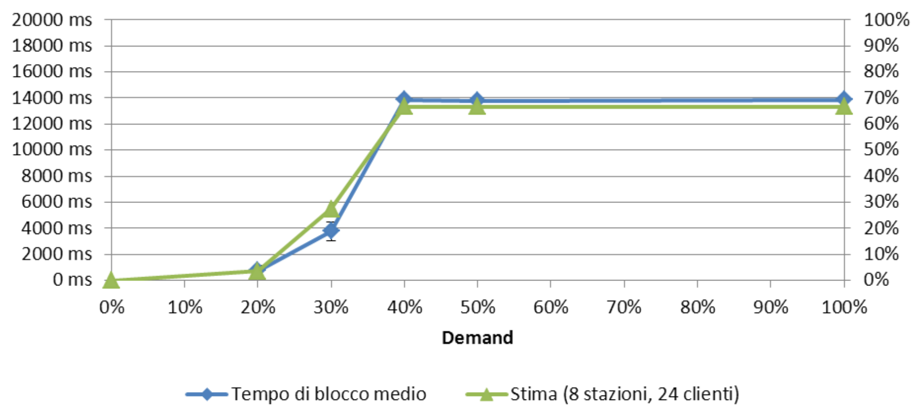
e il tempo di attesa con la formula:

$$\hat{w}^{\%} = 1 - \hat{r}^{\%} - \hat{b}^{\%} \tag{2.10}$$

Questa relazioni riveleranno la loro importanza quando verranno utilizzate per definire degli indici che stimino il servizio offerto dal provider alle singole macchine virtuali.



(a) Stazione cliente 8/20



(b) Stazione cliente 8/24

Figura 2.11: Validazione modello stazione cliente con 5 e 6 macchine virtuali

(a) 3 macchine virtuali (stazione-cliente 8/12)

demand	b	$\hat{b}$	σ	ϵ
0.00%	0.01%	0.00%	0.00%	0.01%
25.00%	0.19%	0.04%	0.04%	0.15%
37.00%	3.05%	0.85%	2.27%	2.20%
43.00%	3.42%	2.58%	2.05%	0.84%
50.00%	10.35%	7.30%	5.31%	3.05%
53.00%	15.52%	10.66%	6.15%	4.86%
56.00%	17.75%	15.02%	4.50%	2.74%
62.00%	23.56%	27.04%	4.83%	3.47%
75.00%	38.50%	33.33%	0.93%	5.17%
100.00%	38.41%	33.33%	1.01%	5.08%

(b) 4 macchine virtuali (stazione-cliente 8/16)

demand	b	$\hat{b}$	σ	ϵ
0.00%	0.01%	0.00%	0.00%	0.01%
25.00%	1.06%	0.75%	0.81%	0.32%
37.00%	7.90%	9.26%	2.97%	1.36%
43.00%	15.81%	20.60%	3.43%	4.78%
50.00%	33.99%	40.18%	5.20%	6.19%
53.00%	49.09%	50.00%	4.31%	0.91%
56.00%	54.11%	50.00%	1.83%	4.11%
62.00%	54.01%	50.00%	0.61%	4.01%
75.00%	53.80%	50.00%	0.80%	3.80%
100.00%	53.83%	50.00%	0.75%	3.83%

Tabella 2.5: Confronto stime e misure del tempo di blocco con 3 e 4 macchine virtuali

(a) 5 macchine virtuali (stazione-cliente 8/20)

demand	b	$\hat{b}$	σ	ϵ
20%	1.04%	1.00%	0.85%	0.05%
30%	7.05%	11.33%	3.60%	4.28%
40%	37.23%	40.44%	6.56%	3.21%
45%	62.97%	60.00%	2.45%	2.97%
50%	63.32%	60.00%	0.70%	3.32%
100%	63.40%	60.00%	0.70%	3.40%

(b) 6 macchine virtuali (stazione-cliente 8/24)

demand	b	$\hat{b}$	σ	ϵ
20%	3.74%	3.62%	0.49%	0.12%
30%	18.86%	27.50%	0.88%	8.64%
40%	69.35%	66.66%	0.21%	2.69%
50%	68.97%	66.66%	0.15%	2.31%
100%	69.34%	66.66%	0.13%	2.68%

Tabella 2.6: Confronto stime e misure del tempo di blocco con 5 e 6 macchine virtuali

Capitolo 3

Metodi per la pianificazione della capacità

Questo capitolo prende in esame la realizzazione di una soluzione in grado di risolvere il problema del dimensionamento di un datacenter dato un aumento previsto nella richiesta di allocazione di macchine virtuali. In particolare, dati l'utilizzo delle macchine virtuali già esistenti e la loro configurazione, si vuole stimare quali, tra quelle nuove, possano essere aggiunte garantendo un livello di servizio, o *SL*, *soddisfacente* e quali, invece, necessitino di nuovi host.

Il vantaggio di questo approccio è la combinazione di informazioni statiche, quali la configurazione delle nuove macchine, e dinamiche riguardanti l'utilizzo corrente dell'hardware. Questa soluzione ha l'obiettivo di trovare una configurazione efficiente che consenta al provider di sfruttare al massimo il proprio datacenter e al cliente di avere un ambiente virtualizzato performante.

Esponendo il problema più formalmente, considerando $\mathcal{VM}$, l'insieme delle macchine virtuali che si prevede richiederanno l'allocazione, si vuole ottenere $\mathcal{VM}_{accepted}$, insieme delle macchine virtuali che possono essere allocate con le risorse del datacenter attualmente disponibili, e $\mathcal{VM}_{newHw}$, insieme di macchine virtuali che richiedono l'acquisto di nuovo hardware per la loro allocazione.

Per poter realizzare questa soluzione si è deciso di creare un algoritmo che nel seguito verrà denominato *Capacity Planning Algorithm* o semplicemente *CPA*. Il *CPA*, oltre ai dati delle macchine virtuali, necessita delle informazioni riguardanti le configurazioni degli host attualmente presenti nel datacenter e del livello di servizio che viene concordato tra provider e cliente.

Lo scopo del *CPA* è anche quello di elaborare l'insieme $\mathcal{H}$, i cui elementi rappresentano gli host, esistenti e da aggiungere, il loro utilizzo e quali macchine virtuali potrebbero ospitare.

Il contenuto delle seguenti sezioni è riassunto qui di seguito.
Nella Sezione 3.1 viene descritto il design ad alto livello dell'algoritmo, soffermandosi sui moduli che compongono la soluzione ed i relativi parametri di input e di output. Per modulo si intende un parte dell'algoritmo che risolve parti del problema generale.

Nella Sezione 3.2 e nella Sezione 3.3 vengono analizzati i due moduli del *CPA*, spiegandone il loro funzionamento interno ed il grado di accuratezza con cui risolvono i problemi loro assegnati.

La Sezione 3.4 propone due indici utili al provider per individuare il livello di performance fornito alle singole macchine virtuali. Questi indici vengono poi integrati nell'algoritmo generale permettendo di valutare l'host di destinazione per le nuove macchine virtuali.

3.1 Panoramica dell'algoritmo

Come anticipato nell'introduzione del capitolo, il *CPA* si compone di due moduli:

Demand Estimator ha il compito di analizzare il tempo di run e il tempo di attesa presenti nei log delle macchine virtuali già allocate negli host e stimarne la *demand*;

Capacity Planner è il cuore della soluzione: una volta ottenuti gli insiemi $\mathcal{VM}$ e $\mathcal{H}$, ha il compito di assegnare ogni macchina virtuale ad uno degli host presenti nel datacenter e, se necessario, stimare il numero di host utili all'allocazione delle macchine che non hanno trovato spazio nella configurazione attuale.

La necessità del *Demand Estimator* nasce dalle osservazioni fatte sugli stress-test. In condizioni di over-commit, infatti, sembra che le informazioni riguardanti la *demand* della singola macchina virtuale vengano parzialmente perse. Ma, poiché, per essere affidabile, un dimensionamento delle risorse necessita della conoscenza dei valori di *demand* di tutte le macchine virtuali, si è deciso di introdurre un componente in grado di stimare nel modo più accurato possibile questo dato.

Basandosi sulle misure del tempo di run e del tempo di attesa campionate in N istanti successivi, il *Demand Estimator* calcola la *demand* istante per istante.

Una volta ottenuto l'andamento temporale della *demand* di tutte le macchine virtuali già allocate, i dati vengono forniti come input al *Capacity Planner*, assieme alle configurazioni delle nuove macchine virtuali. Esso valuta ricorsivamente, attraverso uno o più indici di livello di servizio, la possibilità di inserimento di una nuova macchina virtuale in uno degli host del datacenter.

Le macchine virtuali che non trovano spazio nell'attuale configurazione del datacenter verranno assegnate a nuovi host (insieme $\mathcal{H}_{newHw}$), con un'architettura specificata in input dall'utente, in modo da garantire il livello di servizio come da contratto.

Riassumendo, come mostra la Figura 3.1, il *CPA* richiede in input quattro parametri:

$\mathcal{H}_{available}$ insieme di strutture dati che rappresentano la configurazione degli host disponibili nel datacenter contenenti la variabile *pCpuNum*, numero delle *pCPU* dell'host, ed un array *vms* i cui elementi modellano una macchina virtuale già allocata. In particolare ogni elemento di *vms[]* avrà, oltre alla variabile *vCpuNum*, numero delle *vCPU* della macchina, due array di lunghezza N: *run*, *wait*. I due array conterranno rispettivamente le misure del tempo di run, del tempo di attesa;

$\mathcal{VM}$ insieme di strutture dati che rappresentano la configurazione delle nuove macchine virtuali. Le strutture dati contengono la variabile *vCpuNum*, numero delle *vCPU* della macchina modellata;

SL[] array che contiene uno o più parametri numerici compresi tra 0 e 1 che indicano il livello di servizio pattuito tra provider e cliente. Un SL ≈ 1 indica un livello di performance del servizio massimo, al contrario un SL ≈ 0 non dà alcuna garanzia al cliente. La presenza di indici multipli permette all'algoritmo di valutare diversi parametri di performance;

HostCfg struttura dati contente la variabile *pCpuNum* utile alla configurazione dei nuovi host eventualmente necessari.

L'output generato dal *CPA* è composto da quattro parametri:

$\mathcal{VM}_{accepted}$ insieme di strutture dati che contengono le informazioni stimate delle singole macchine virtuali ed il loro livello di servizio offerto, allocate con il sistema esistente;

$\mathcal{VM}_{newHw}$ insieme di strutture dati che contengono le informazioni delle macchine virtuali che necessitano di nuovi host per poter essere allocate;

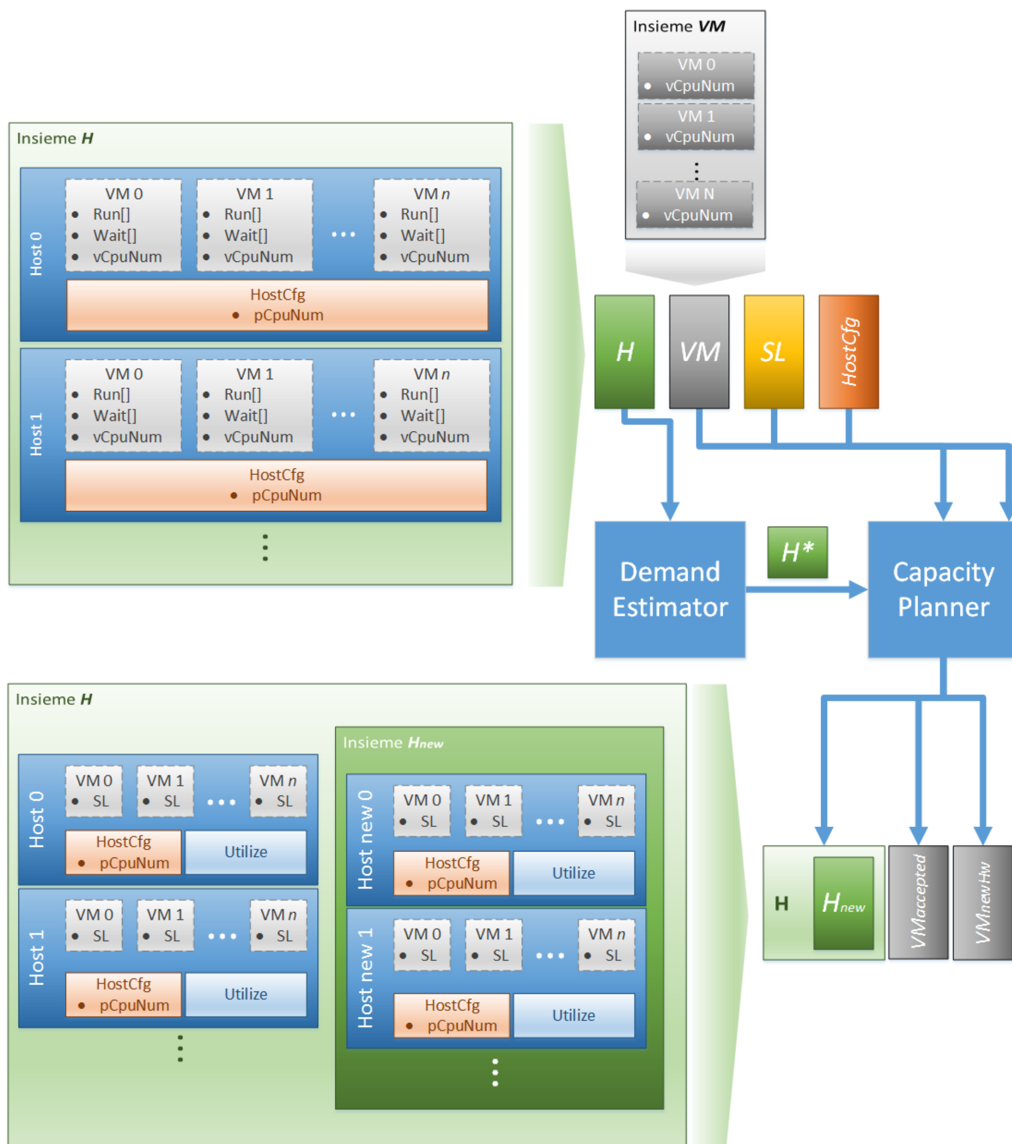


Figura 3.1: Panoramica del *Capacity Planning Algorithm*

$\mathcal{H}$ insieme di strutture dati contenenti le informazioni di tutti gli host (nuovi e già esistenti): utilizzo stimato, numero di *pCPU* e macchine virtuali allocate.

$\mathcal{H}_{new}$ è un sottoinsieme di $\mathcal{H}$ che contiene gli elementi che rappresentano solo gli host da aggiungere.

Nella sezione successiva si analizza nel dettaglio la parte di algoritmo riguardante il *Capacity Planner*. Una volta compreso il suo funzionamento sarà evidente la necessità di una scelta opportuna degli indici che valutino il livello di servizio offerto e del ruolo fondamentale svolto dal *Demand Estimator*.

3.2 Capacity Planner

Per poter individuare gli insiemi $\mathcal{VM}_{accepted}$ e $\mathcal{VM}_{newHw}$ viene eseguito un processo iterativo che simula l'inclusione una ad una delle nuove macchine virtuali nel sistema. Se il *SL* non può essere garantito dal datacenter la macchina virtuale viene inserita in un nuovo host.

L'algoritmo riceve in input l'insieme $\mathcal{VM}$, il *SL* da soddisfare, *HostCfg*, la configurazione che devono avere i nuovi host, e l'insieme $\mathcal{H}$, i cui elementi contengono gli host disponibili e le macchine virtuali allocate con il loro utilizzo e la loro *demand* stimata.

Ad ogni iterazione una macchina virtuale viene estratta dall'insieme $\mathcal{VM}$ e assegnata ad un host, finchè l'insieme non risulta vuoto. L'ordine con cui vengono estratte le macchine virtuali è casuale (linea 6 dell'Algoritmo 1); più precisamente, ogni macchina virtuale ha probabilità $|\mathcal{VM}|^{-1}$ di essere selezionata in un iterazione. Questa scelta vuole simulare l'assenza di conoscenza riguardo l'ordine con cui avverranno le richieste di allocazione. Ciò, inoltre, evita di ottenere un output troppo ottimistico, selezionando prima le macchine virtuali con una capacità maggiore, o troppo pessimistico, aggiungendo al datacenter prima le macchine virtuali con una capacità virtuale minore.

La funzione principale contenuta nell'algoritmo è la *getHostForVm*. Questa funzione simula la richiesta di inclusione di una macchina virtuale che fa richiesta di tutte le sue risorse computazionali virtualizzate. Il *Capacity Planner* stabilisce se l'host può o meno ospitare la macchina virtuale calcolando il *SL* della nuova macchina virtuale e di quelle già presenti e valutando che ognuno di essi sia maggiore o uguale al valore di *SL* ricevuto in input. Per fare ciò, l'algoritmo fa uso della *demand* stimata delle macchine virtuali esistenti.

Algoritmo 1 Dimensionamento del datacenter

Require: $\mathcal{VM} \neq \emptyset$
and vm.vCpuNum $\leq$ HostCfg.pCpuNum
$\forall$ vm $\in \mathcal{VM}_{new}$
and SL.lenght > 0

Ensure: vm.sl[i] $\geq$ SL[i]
$\forall$ h $\in \mathcal{H}$ ($\forall$ vm $\in$ h.vms[]) and $\forall$ i $\in$ [0, SL.lenght - 1]

```
1: H_newHw := ∅
2: H_new := ∅
3: VM ← VM_running ∪ VM_new
4: H ← H_available
5: while VM ≠ ∅ do
6:     vm ← getAndRemoveRandomItem(VM)
7:     host ← getHostForVM(vm, H, SL)
8:     if host == null then
9:         host ← createNewHost(HostCfg)
10:        addItem(H_new, host)
11:        addItem(H, host)
12:    end if
13:    addItem(VM_allocated, vm)
14:    addVmToHost(host, vm)
15:    host.utilize[] ← updateHostUtilize(host)
16: end while
17: for all host ∈ H_new do
18:    addAllItem(VM_newHw, host.vms)
19: end for
20: VM_accepted ← VM_allocated \ VM_newHw
21: return VM_accepted, VM_newHw, H, H_new
```

Se uno degli host riesce a rispettare il livello di servizio richiesto, la macchina virtuale può essere allocata nel sistema. In questo caso *getHostForVm* restituisce l'host dove la macchina può essere allocata (linea 7 dell'Algoritmo 1). Se invece nessun host è in grado di ospitare la macchina virtuale, la *getHostForVm* restituisce un valore *null* e viene attivato l'*if-statement* (linea 8 dell'Algoritmo 1) che simula l'aggiunta di un nuovo host al datacenter. Tramite la funzione *createNewHost*, l'algoritmo crea un nuovo host che viene aggiunto all'insieme $\mathcal{H}$ e all'insieme $\mathcal{H}_{new}$. L'insieme $\mathcal{H}_{new}$ sarà utile, una volta terminato l'algoritmo, per tener traccia dell'hardware nuovo, necessario a soddisfare tutte le richieste di allocazione.

Per ogni host h appartenente all'insieme $\mathcal{H}$, vengono salvate nell'array *h.vms[]* le informazioni delle macchine virtuali allocate. Il numero di *vCPU* della macchina virtuale si trova, invece, nella variabile *h.vms[i].vCpuNum*

con i, l'indice che identifica la macchina.
Nell'array *h.vms[i].demand[]* di lunghezza N, numero di campionamenti nel log, viene salvata l'informazione sulla *demand* stimata istante per istante.

Terminata l'associazione di tutte le macchine virtuali ad un host, l'algoritmo termina creando l'insieme $\mathcal{VM}_{newHw}$, selezionando le macchine virtuali che non hanno trovato spazio negli host del datacenter attuale (linee 17-19 dell'Algoritmo 1), e con la creazione dell'insieme $\mathcal{VM}_{accepted}$ (linea 20 dell'Algoritmo 1) risultante dalla differenza tra tutte le macchine virtuali allocate e l'insieme $\mathcal{VM}_{newHw}$.

Da notare che l'algoritmo proposto non elabora un'allocazione ottimale delle risorse. Le caratteristiche dell'i-esima nuova macchina virtuale, infatti, non possono essere predette ed un bilanciamento degli host tramite la migrazione di alcune macchine virtuali non è considerata come una soluzione vantaggiosa da introdurre in questo punto della soluzione.

Per il corretto funzionamento dell'algoritmo sono necessarie le seguenti pre-condizioni:

- l'insieme $\mathcal{VM}$ non deve essere vuoto;
- le macchine virtuali dell'insieme $\mathcal{VM}$ devono avere un numero di *vCPU* minore o uguale al numero di *pCPU* degli eventuali nuovi host, pena l'impossibilità di allocazione;
- deve esistere almeno un valore di *SL*.

Al termine dell'esecuzione l'algoritmo assicura la seguente post-condizione:

- tutte le macchine virtuali, nuove e già allocate, avranno un livello di servizio maggiore o uguale a quello garantito.

3.3 Demand Estimator

Una volta compreso il funzionamento del *Capacity Planner* è immediato comprendere l'utilità di un componente che, date le misure disponibili a livello dell'hypervisor, sia in grado di stimare quale sia la *demand* delle macchine virtuali attualmente già allocate. Come evidenziato già da altri studi [13], una dei maggiori limiti dovuti all'osservazione esterna delle macchine virtuali è l'impossibilità di misurare la reale richiesta di risorse nel momento in cui l'host è completamente utilizzato.

Questa sezione propone una soluzione per profilare la *demand* attraverso l'utilizzo dell'informazione di N campionamenti successivi del valore medio calcolato su un tempo t^{sample} dei parametri per *vCPU* di *Run* e *Wait*,

parametri che si possono monitorare attraverso i tool di sistema. In questo modo è possibile ottenere un istogramma che approssimi la funzione di distribuzione che descrive la *demand*.

La difficoltà nella predizione della *demand* risulta evidente nei casi modellati in Figura 3.2 che mostra i tempi di *Run*, *Block* e *Wait* registrati da due macchine virtuali single-core con *demand* pari al 50% allocate su un host con una sola *pCPU* in un arco di tempo t.

Se le richieste di utilizzo della *pCPU* sono simultanee (caso in Figura 3.2(a)) lo scheduler dell'host alternerà gli istanti d'esecuzione tra le due macchine virtuali facendo registrare un tempo di run pari a $0.5t$ (come da $demand[]$), un tempo di blocco di $\approx 0.5t$ e un tempo di attesa di $\approx 0t$.

Di contro nell'esempio in Figura 3.2(b), la seconda richiesta di utilizzo arriva esattamente dopo un tempo $0.5t$, quando, cioè, la prima macchina virtuale ha completato il suo carico di lavoro. Questo fa sì che l'host possa soddisfare la richiesta con un tempo di run pari a $0.5t$, tempo di blocco di $0t$ e tempo di attesa i $0.5t$.

Tuttavia il tempo di run non è sempre un indicatore affidabile della *demand*. Per capire ciò è sufficiente considerare l'esempio in Figura 3.3 e confrontarlo con quello in Figura 3.2. Sebbene la Figura 3.3 mostri dei valori di tempo di run, tempo di blocco e tempo di attesa uguali a quelli del caso in Figura 3.2, questa volta il valore della *demand* di entrambe le macchine virtuali è pari all'80%. La necessità di gestire l'over-commit, però, porta il sistema ad assegnare un utilizzo della *pCPU* con un valore di *Run* pari al 50%, facendo registrare un tempo di blocco di $0.5t$ ed un tempo di attesa di $0t$.

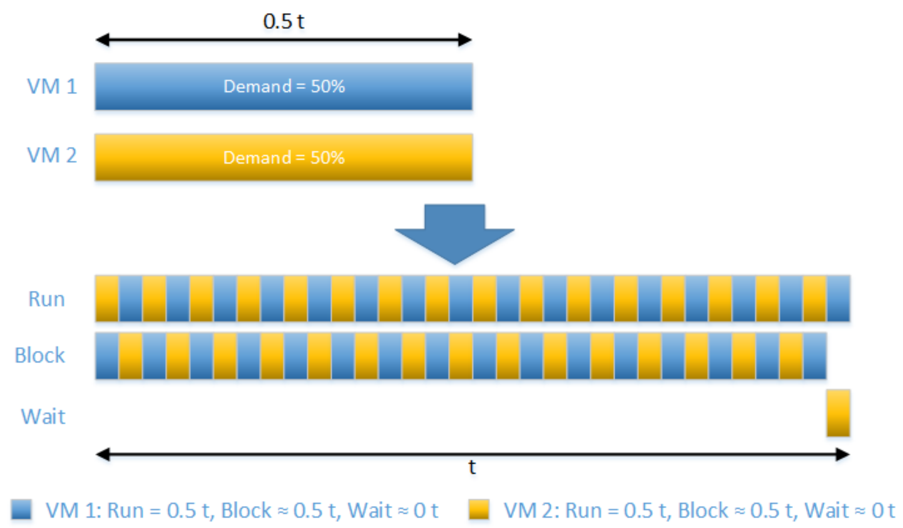
In questo secondo caso l'informazione sul valore della *demand* si perde quasi completamente. Infatti, poiché entrambe le macchine virtuali registrano un tempo di blocco pari allo $0.5t$, si può solo affermare che stiano richiedendo un utilizzo della risorsa fisica uguale o superiore al 50%.

Facendo un campionamento della media del tempo di attesa in un intervallo *sufficientemente* ampio, come può essere quello fornito dallo strumento di monitoring di vSphere Client pari a $20s$, è ragionevole supporre che il valore di tempo di attesa misurato nel caso in Figura 3.2, appartenente all'intervallo $[0, 0.5t]$, sia verosimilmente $\gg 0$.

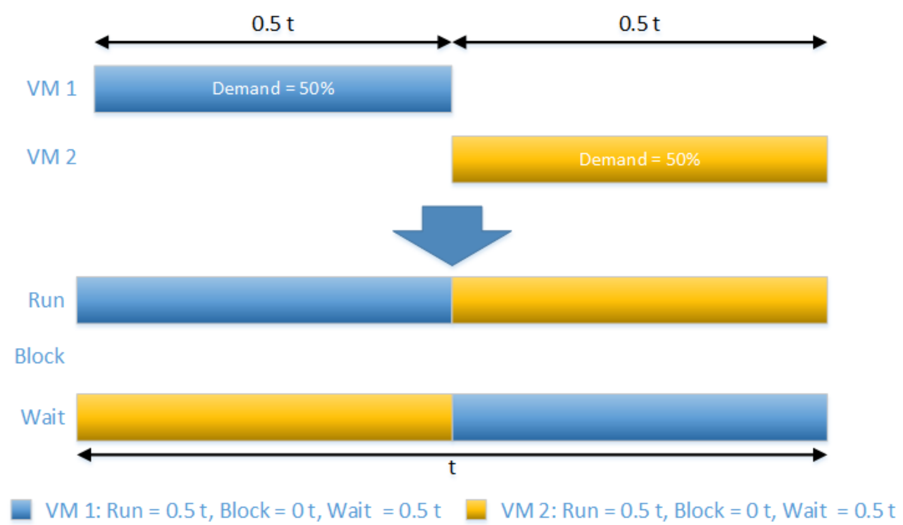
Nel caso in Figura 3.3, invece, la media del tempo di attesa avrà dei valori sempre molto *piccoli* ≈ 0.

Da queste osservazioni ne derivano le seguenti relazioni:

$$\begin{aligned} w^{\%} \gg 0 \quad &\Leftrightarrow \quad d^{\%} = r^{\%} \\ w^{\%} \approx 0 \quad &\Leftrightarrow \quad d^{\%} \geq r^{\%} \end{aligned}$$



(a)



(b)

Figura 3.2: Esempio di due macchine virtuali con stessa *demand* con valori di *Run*, *Block* e *Wait* diversi

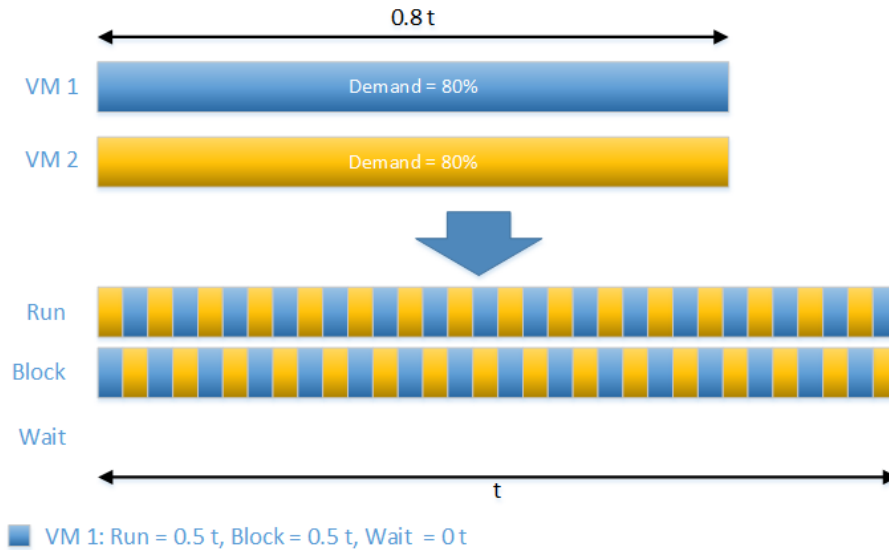


Figura 3.3: Esempio con *demand* diversa da quella in Figura 3.2 ma con stessi valori misurati

In conclusione in stato di over-commit non è possibile fare una stima accurata della *demand*. È possibile, però, definire alcune caratteristiche statistiche creando un istogramma che aggreghi i valori della *demand* e aggiornarlo in maniera dinamica con dati che incrementino l'accuratezza dello stesso, utilizzando tecniche di gestione dell'incertezza.

3.3.1 Calcolo dell'istogramma della *demand*

L'istogramma della *demand* di una macchina virtuale rappresenta la frequenza di una richiesta di risorse entro un certo intervallo. Dato K il numero di intervalli che compongono l'istogramma, la larghezza di ogni intervallo è pari a $\frac{1}{K}$ (es. il dominio della *demand* può essere composto da $K = 4$ intervalli i_k ed ognuno di essi rappresenta un intervallo del 25%: $i_1 = (0\% - 25\%)$, $i_2 = (25\% - 50\%)$, $i_3 = (50\% - 75\%)$, $i_4 = (75\% - 100\%)$).

Per la creazione dell'istogramma, insieme degli intervalli i_k, dato un certo istante n (indicato da qui in avanti con I_K^n), si sfruttano i dati già raccolti all'istante $n-1$, presenti nell'istogramma I_K^{n-1}, oltre ad un metodo noto come *principle of insufficient reason* per la gestione dell'incertezza per gli stati di over-commit.

Nell'instante zero l'istogramma è vuoto: tutti i K intervalli hanno valore uguale a 0. Gli istanti successivi, invece, sono regolati da una semplice relazione: la somma di tutti i valori nell'istante n è maggiore di un'unità

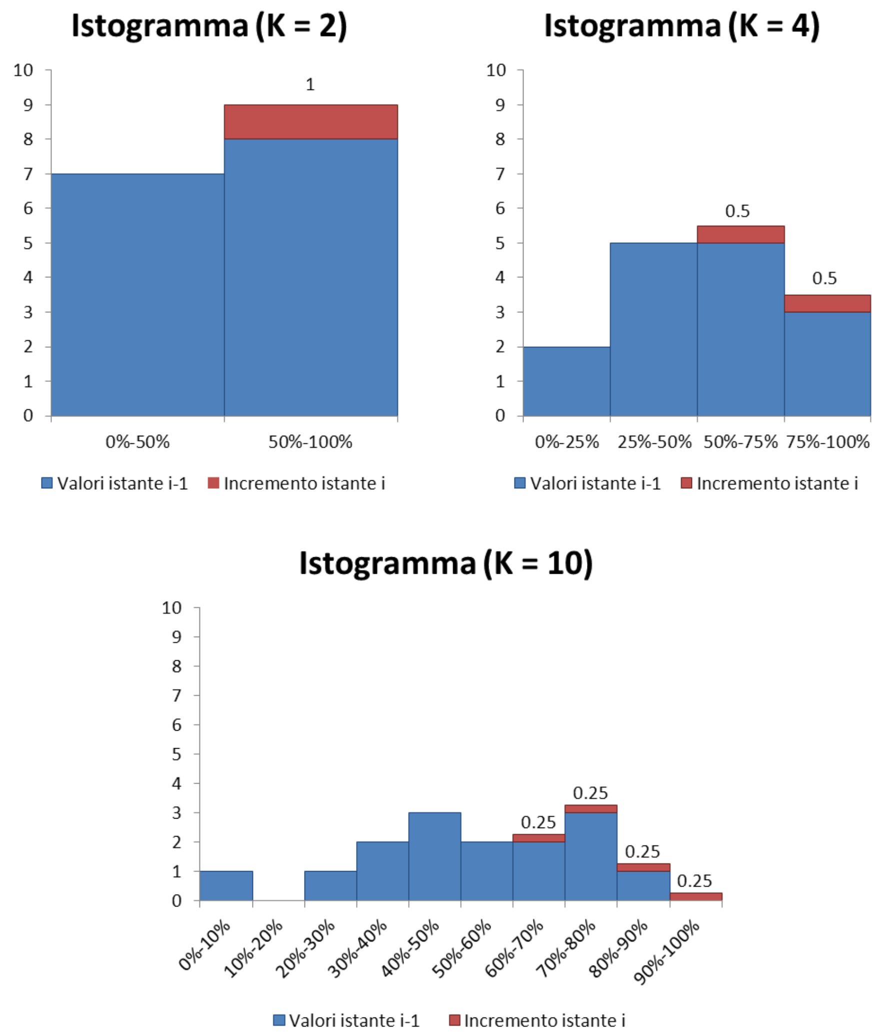


Figura 3.4: Esempi di incremento dell'istogramma con K=2, K=4, K=10 con $r^{\%} = 65\%$ e $w^{\%} \approx 0\%$

rispetto alla somma dei valori nell'istante $n-1$:

$$\sum_{k=0}^{K} i_k^n = 1 + \sum_{k=0}^{K} i_k^{n-1}$$

La creazione dell'istogramma della *demand* di una macchina virtuale avviene considerando istante per istante il valore $r^{\%}$. Se il valore di $w^{\%}$ è considerato accettabile, l'intervallo dell'istogramma che rappresenta la *demand* $d^{\%} = r^{\%}$ viene incrementato di 1. Se invece il valore di $w^{\%}$ è ≈ 0 l'unità di incremento viene suddivisa, proporzionalmente, su tutti gli intervalli che rappresentano valori $d^{\%} \geq r^{\%}$.

In Figura 3.4 viene evidenziato l'incremento di tre istogrammi, rispettivamente con $K = 2$, $K = 4$ e $K = 10$, nel caso in cui venga misurato un valore di $r^{\%} = 65\%$ e un valore di $w^{\%} \approx 0$.

3.3.2 Stima della *demand*

Una volta realizzato l'istogramma della *demand* con tutti i valori presenti nel log della macchina virtuale, è possibile dare una stima della *demand* complessiva riportando i valori di $r^{\%}$ laddove $w^{\%} \gg 0$. Quando $w^{\%} \approx 0$, $\hat{d}$ è un campionamento casuale ottenuto dagli intervalli dell'istogramma della *demand* che rappresentano $d^{\%} \geq r^{\%}$. Da alcuni test realizzati un intervallo del tempo di attesa accettabile per considerare lo stato della macchina in over-commit è $0\% \leq w^{\%} < 1\%$.

L'Algoritmo 2 mostra la creazione dell'istogramma della *demand* di una macchina virtuale. L'algoritmo necessita, per l'appunto, degli array *run[]*, *wait[]*, che devono essere di uguale lunghezza, e di K, numero degli intervalli in cui suddividere l'istogramma.

L'algoritmo iterativo valuta ad ogni esecuzione se il valore di *Run* misurato può essere considerato pari al valore della *demand*. In caso affermativo viene incrementato di un'unità solo l'intervallo corrispondente, altrimenti viene calcolato un valore δ da suddividere tra gli intervalli che identificano una *demand* maggiore o uguale al valore di *Run* misurato.

Più in particolare, l'*if-statement* della linea 3 dell'Algoritmo 2 individua i valori per cui si ritiene la macchia virtuale in stato di over-commit e, nel caso, anziché incrementare l'intervallo corrispondente di una unità (linea 4 dell'Algoritmo 2), suddivide l'incremento su più intervalli (linee 6-9 dell'Algoritmo 2).

L'Algoritmo 3, invece, mostra l'effettivo funzionamento del *Demand Estimator* che, sfruttando l'istogramma precedentemente creato, genera un array *demand[]* di lunghezza pari a quella dell'array *run[]*, campionando, quan-

do necessario, i valori dall'istogramma tramite la funzione *getRandomValue-GEQ* (linea 5 dell'Algoritmo 3).

Algoritmo 2 Calcolo dell'istogramma della *demand*

Require: run.lenght == wait.lenght and $K > 0$
Ensure: $\sum_{k=1}^{K}$ histogram[k] == run.lenght
1: **for** i= 1 to run.lenght **do**
2: k $\leftarrow \left\lceil \frac{run[i]}{100} \cdot K \right\rceil$
3: **if** wait[i]> 1 **then**
4: histogram[k] ← histogram[k] +1
5: **else**
6: $\delta \leftarrow \frac{1}{K+1-k}$
7: **for** j=k to K **do**
8: histogram[j] ← histogram[j] +δ
9: **end for**
10: **end if**
11: **end for**
12: **return** histogram

Algoritmo 3 Stima della *demand* di una macchina virtuale

Require: run.lenght == wait.lenght
Ensure: run[i] ≤ demand[i]
∀ i ∈ [0, run.lenght - 1]
1: **for** i= 1 to N **do**
2: **if** wait[i]> 1 **then**
3: demand[i] ← run[i]
4: **else**
5: demand[i] ← getRandomValueGEQ(histogram, run[i])
6: **end if**
7: **end for**
8: **return** demand

3.4 Indici del livello di servizio

Come spiegato nella Sezione 3.2, il *Capacity Planner* necessita di almeno un indice che valuti il livello di performance offerto per ogni macchina virtuale. In questa sezione vengono proposti due tipi di indici che permettano di valutare il livello di servizio offerto. Prima di vedere come questi sono calcolati è importante, però, dare una definizione che sarà utile anche nei capitoli successivi.

Un indice di livello di servizio viene indicato con *sl*. Gli indici proposti sono funzioni che hanno come parametri il numero di *pCPU* dell'host e le coppie *demand*, numero di *vCPU* delle macchine virtuali che fanno richiesta di utilizzare la risorsa. Generalizzando si può definire formalmente l'indice di livello di servizio come

$$sl(\mathcal{D}^t, N_h^{pCPU})$$

dove

$\mathcal{D}^t$ insieme di coppie $(d_v^{\%}(t), N_v^{vCPU})$;

$d_v^{\%}(t)$ valore *demand* della macchina virtuale v ad un certo istante di tempo t;

$N_v^{\boldsymbol{vCPU}}$ numero delle *vCPU* della macchina virtuale v;

$N_h^{\boldsymbol{pCPU}}$ numero delle *pCPU* dell'host h.

Indice di Response Time

Il primo indice è definito come il rapporto tra il tempo di run e la somma del tempo di run con il tempo di blocco.

$$sl^{RT} = \frac{t^{Run}}{t^{Run} + t^{Block}} = \frac{r^{\%}}{r^{\%} + b^{\%}}$$

L'utilizzo dell'insieme $\mathcal{D}^t$ e di N_h^{pCPU} è insito nei valori di $r^{\%}$ e di $b^{\%}$.

Questo indice, chiamato nel seguito *indice di Response Time* o semplicemente sl^{RT}, sfrutta due concetti che nella teoria delle code[3] sono noti con i termini di *Service Time* e *Response Time*. Il *Service Time* indica il tempo in cui il cliente utilizza effettivamente la stazione. Il *Response Time* è la somma del *Service Time* e del tempo trascorso in attesa, prima, cioè, di poter usare la stazione.

La necessità di questo indice nasce dal fatto che anche in un ambiente di virtualizzazione ideale, come quello modellato nel Capitolo 2, nel momento in cui il numero di *vCPU* supera il numero di *pCPU*, il tempo di blocco assume valori > 0 dovuti alla richiesta contemporanea di due o più *vCPU*.

Trascurare questo fatto porterebbe a pensare che un host possa accettare un numero illimitato di macchine virtuali purché la *demand* totale sia inferiore a quella che è in grado di fornire.

I test illustrati nella Sezione 2.3.2 del Capitolo 2, invece, hanno mostrato come la presenza di *troppe* macchine virtuali su un unico host portano a

valori del tempo di blocco non trascurabili anche nel caso di bassi valori di *demand*.

Un valore di $sl^{RT} \approx 1$ evidenzia lo stato di una macchina virtuale che ottiene l'utilizzo della CPU fisica non appena ne fa richiesta. Di contro un valore ≈ 0 è sintomo di una macchina virtuale che è permanente in attesa di utilizzare la CPU.

Indice di Troughtput

L'indice sl^{RT} non utilizzando il valore della *demand*, non è in grado di distinguere lo stato di over-commit dallo stato normale. In altre parole l'indice sl^{RT} di una macchina virtuale con $d^{\%} = 40\%$, $r^{\%} = 40\%$ e $b^{\%} = 10\%$ (stato normale) è pari all'indice sl^{RT} di una seconda macchina virtuale con $d^{\%} = 100\%$, $r^{\%} = 80\%$ e $b^{\%} = 20\%$ (stato di over-commit). Se nel primo caso la macchina virtuale riesce a completare il suo carico di lavoro, seppure con un certo tempo di ritardo, nel secondo caso la macchina virtuale, oltre al ritardo, percepisce anche un calo della frequenza della CPU del 20%.

A causa di questa limitazione, si è deciso di introdurre un secondo indice che metta in relazione la *demand* $d^{\%}$ della macchina virtuale con il tempo di run in percentuale $r^{\%}$. Questo indice, chiamato nel seguito *indice di Throughtput* o semplicemente sl^{T}, è definito come

$$sl^{T} = \frac{r^{\%}}{d^{\%}}$$

Come intuibile l'indice sl^{T}, sfrutta il concetto di Throughtput che nella teoria della code indica quanto *lavoro* offre la stazione nel singolo istante di tempo. In questo contesto una riduzione di $d^{\%}$ a $r^{\%}$ porta ad una diminuzione del Throughtput pari a $(d^{\%} - r^{\%}) \cdot \nu$ con ν la frequenza nominale della CPU. In maniera più intuitiva l'indice sl^{T} indica la capacità di calcolo fornita alla macchina virtuale, rapportata alla *demand* della stessa.

3.4.1 Semplificazione dell'indice di *Response Time*

Se per il calcolo dell'indice di Throughtput, una volta stimata la *demand*, non vi sono grosse difficoltà nel calcolare il tempo di run (è sufficiente infatti applicare l'algoritmo di scheduling illustrato nella Sezione 1.4 del Capitolo 1), il calcolo dell'indice di *Response Time*, che necessita della stima del tempo di blocco, presenta notevoli complicazioni. Attraverso i test presentati nel Capitolo 2, infatti, si è riusciti ad avere la relazione illustrata dall'Equazione 2.8 che stima con buona approssimazione il tempo di blocco nel caso in cui le macchine virtuali facciano la stessa richiesta di utilizzo.

Nel caso in cui, invece, le macchine virtuali abbiano una *demand* diversa tra di loro, la relazione trovata non è più valida.

Si è quindi deciso di effettuare una semplificazione nel calcolo del tempo di blocco e quindi dell'indice di *Response Time*: a tutte le macchine virtuali viene assegnato un indice sl^{RT} pari a

$$sl^{RT} = \frac{\bar{r}^{\%}}{\bar{r}^{\%} + \bar{b}^{\%}}$$

con

$\bar{r}^{\%}$ percentuale del tempo di run ottenuto dall'Equazione 2.9 considerando la *demand* pari a $\bar{d}^{\%}$, media dei valori di *demand* reali;

$\bar{b}^{\%}$ percentuale del tempo di blocco ottenuto dall'Equazione 2.8 considerando la *demand* pari a $\bar{d}^{\%}$, media dei valori di *demand* reali.

Pur non rispecchiando la reale situazione delle singole macchine virtuali, il valore di tempo di blocco calcolato rappresenta il limite superiore che può essere misurato negli stati in cui la media della *demand* rimane uguale a quella dello stato in oggetto.

La dimostrazione formale di questo concetto esula dagli scopi di questo elaborato. Ad ogni modo per convincersi è sufficiente abbozzare un ragionamento per induzione utilizzando un piccolo esempio. Il primo evidenzierà l'ipotesi in una situazione standard, non in stato di overcommit. Il secondo, invece, mostrerà l'evidenza dell'ipotesi in stato di overcommit.

Primo esempio Supponiamo di avere 1 *pCPU* e 2 *vCPU* che hanno una *demand* rispettivamente dell'80% e del 10%. La probabilità che entrambe facciano richiesta della *pCPU* è pari a $0.8 \cdot 0.1 = 0.08 = 8\%$.

Considerando invece la media della *demand* pari al 45%, la probabilità risulta essere $0.45 \cdot 0.45 = 0.2025 = 20.25\%$, superiore a quella calcolata nello stato reale.

Secondo esempio Questa volta supponiamo di avere sempre 1 *pCPU* e 2 *vCPU* ma che hanno una *demand* rispettivamente dell'80% e del 40%. In questo caso la probabilità che entrambe le *vCPU* richiedano l'unità fisica è pari a $0.8 \cdot 0.4 = 0.32 = 32\%$.

Considerando invece la media della *demand* pari al 60%, la probabilità risulta essere $0.6 \cdot 0.6 = 0.36 = 36\%$, superiore a quella calcolata nello stato reale.

In conclusione operare questa semplificazione per il calcolo dell'indice di *Response Time* consente di valutare l'indice nel caso peggiore, dando al provider un valore comunque significativo sullo stato attuale dell'host.

3.4.2 Livello di servizio aggregato

In aggiunta alla definizione degli indici di livello di servizio è necessario introdurre un metodo per aggregare i valori stimati istante per istante.

Questa necessità nasce dal fatto che, per la valutazione complessiva della performance misurata, è importante rilevare un livello di servizio alto nel lungo periodo e non in ogni singolo istante.

Infatti, come caso limite, si può considerare un host che ad un certo istante misura un picco improvviso della *demand* delle macchine virtualizzate, picco che si esaurisce in un tempo *sufficientemente* breve. In questo istante l'host misurerebbe un livello di servizio al di sotto del limite tollerato. Il sistema, pur riuscendo a garantire nel complesso un livello di servizio più che accettabile, a causa di quel unico picco, impedirebbe l'aggiunta di altre macchine virtuali.

Poiché dare un limite statico da rispettare in ogni istante di tempo risulta una soluzione troppo drastica, si vuole, in questa sezione, proporre un metodo che riesca garantire una certa tolleranza in caso di inadempienza del livello di servizio minimo. L'idea di base è quella di aggregare una serie di valori $sl(1), sl(2), \cdots, sl(N-1), sl(N)$ in un unico valore, definito come indice di livello di servizio aggregato (o *Aggregated Service Level*), asl.

Una prima soluzione può essere quella di calcolare la media aritmetica di tutti gli indici calcolati. Quindi, risulta che

$$asl = \frac{\sum\limits_{n=1}^{N} sl(n)}{N} \tag{3.1}$$

L'Equazione 3.1, che rappresenta il livello di servizio medio, identifica un parametro solitamente usato in letteratura. Per esigenze industriali, però, si è considerata la possibilità di penalizzare maggiormente gli istanti in cui il livello di servizio raggiunge livelli troppo bassi.

In altre parole un livello di servizio aggregato pari a 0.7 può essere ottenuto sia da due istanti con $sl(1) = 0.4$ e $sl(2) = 1$, sia da due istanti con $sl(1) = 0.6$ e $sl(2) = 0.8$. Pur non raggiungendo mai la condizione ottimale, il secondo scenario risulta più favorevole al cliente in quanto non misura mai un calo drastico delle performance come invece accade nel primo.

Per poter valutare in maniera differente le due situazioni sopra descritte, si è scelto di introdurre il concetto di livello di servizio percepito (o *Perceived Service Level*), psl. Il livello di servizio percepito viene definito come

$$psl = 1 - penalty(sl) \tag{3.2}$$

dove $penalty(sl)$ è una funzione che, dato il livello di servizio, restituisce un valore di penalità maggiore o uguale a 0. Una volta noto il $psl(t)$ per ogni istante di tempo, è possibile effettuare la media aritmetica dei valori ottenendo il livello di servizio aggregato come

$$asl = \frac{\sum_{n=1}^{N} psl(n)}{N} \tag{3.3}$$

Con questa definizione l'Equazione 3.1 può essere vista anche come media dei valori di livello di servizio percepito la cui funzione di penalità, o semplicemente penalty, risulta:

$$penalty^{lin}(sl) = 1 - sl \tag{3.4}$$

da cui, secondo l'Equazione 3.2, si ha

$$psl^{lin} = 1 - penalty^{lin}(sl) = sl$$

La funzione di penalità dell'Equazione 3.4 viene definita come penalty lineare, in quanto il valore di penalità viene incrementato linearmente rispetto al calo del livello di servizio.

Questa definizione ha permesso di introdurre una seconda funzione di penalità che nasce dall'idea di rendere impossibile raggiungere un livello di servizio pari a 0. Questo viene tradotto matematicamente con una penalty infinita quando $sl = 0$.
Una possibile funzione di penalità che è stata ritenuta di immediata applicazione è la seguente:

$$penalty^{asy}(sl) = \frac{1}{sl} - 1 \tag{3.5}$$

L'Equazione 3.5, ottenuta partendo dall'equazione dell'iperbole equilatera diminuita di un unità, ha un valore di penalità uguale a 0 se $sl = 1$ e tende a ∞ se $sl \to 0$.

Nota, questa funzione di penalità viene definita in seguito come penalty asintotica.

Dalla Equazione 3.2 si ottiene che

$$psl^{asy} = 1 - penalty^{asy}(sl) = 2 - \frac{1}{sl}$$

Il grafico in Figura 3.5 mette a confronto il livello di servizio percepito in funzione del livello di servizio, ottenuto con penalty lineare (linea rossa) e con penalty asintotica (linea verde). Si può notare come la penalty asintotica

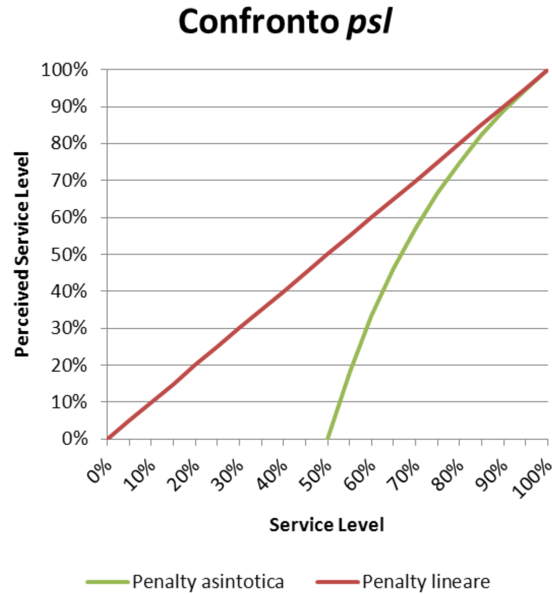


Figura 3.5: Confronto del livello di servizio percepito con penalty lineare (linea rossa) e con penalty asintotica (linea verde)

fa sì che il livello di servizio cali in maniera molto più rapida, rispetto a quello ottenuto con la penalty lineare. Non solo, quando $sl < 0.5$, il livello di servizio percepito assume anche valori negativi.

In questo modo, applicando l'Equazione 3.3, quando il livello del servizio assume valori *bassi*, ha un peso maggiore nella media aritmetica utilizzata per calcolare il livello di servizio aggregato.

Le Tabelle 3.1 e 3.2 mostrano un confronto del livello di servizio aggregato con penalty lineare e asintotica, negli scenari presentati ad inizio sezione. Nello Scenario 1 (Tabella 3.1) vengono misurati due livelli di servizio pari a $sl(1) = 0.4$ e $sl(2) = 1$. Nello Scenario 2 (Tabella 3.2) vengono misurati due livelli di servizio pari a $sl(1) = 0.6$ e $sl(2) = 0.8$.

Come mostrano i dati, il livello di servizio aggregato in caso di penalty lineare è uguale per entrambi gli scenari. Se, invece, si utilizza la penalty asintotica lo Scenario 2 risulta più performante rispetto al primo.

Scenario 1	$t = 1$	$t = 2$	asl
sl	0.4	1	0.7
psl^{lin}	0.4	1	0.7
psl^{asy}	-0.5	1	0.25

Tabella 3.1: Confronto del livello del servizio aggregato con $sl(1) = 0.4$ e $sl(2) = 1$

Scenario 1	$t = 1$	$t = 2$	asl
sl	0.6	0.8	0.7
psl^{lin}	0.6	0.8	0.7
psl^{asy}	0.33	0.75	0.54

Tabella 3.2: Confronto del livello del servizio aggregato con $sl(1) = 0.6$ e $sl(2) = 0.8$

In ultima analisi si può notare, però, che la funzione di penalità asintotica, se da un lato consente di dare una maggiore peso ai picchi negativi, non dà alcuna rilevanza agli istanti in cui il livello di servizio viene rispettato. Se, per esempio, nello Scenario 2 in Tabella 3.2, $sl \geq 0.7$ fosse stato un valore di livello di servizio accettabile, sarebbe stato opportuno che per gli istanti in cui $sl \geq 0.7$, il livello di servizio percepito fosse $psl = sl$.

Partendo da questa osservazione si è deciso di introdurre una terza funzione di penalità che combina le due scelte prima. Se il livello di servizio misurato è al di sopra di un certo parametro sl^*, la penalty mantiene la forma lineare. Se, invece, si trova al disotto del valore di sl^*, la penalty viene calcolata secondo una funzione asintotica. La penalty scelta ha questa forma:

$$penalty^{mix}(sl) = \begin{cases} 1 - sl & sl \geq sl^* \\ \dfrac{a}{sl} + c & sl < sl^* \end{cases} \tag{3.6}$$

I parametri a e c vengono calcolati in base al valore limite sl^* secondo le condizioni di derivabilità e continuità nel punto $sl = sl^*$.

In sintesi, per la condizione di derivabilità, si ha che

$$a = sl^{*2}$$

Mentre, per la condizione di continuità, si ha che

$$c = 1 - sl^* - \frac{a}{sl^*} = 1 - 2sl^*$$

Sostituendo i parametri a e c nell'Equazione 3.6 e applicando l'Equazione 3.2 si ottiene

$$psl^{mix} = 1 - penalty^{mix}(sl) = \begin{cases} sl & sl \geq sl^* \\ 1 - \left(\dfrac{sl^{*2}}{sl} + 1 - 2 \cdot sl^*\right) = 2 \cdot sl^* - \dfrac{sl^{*2}}{sl} & sl < sl^* \end{cases}$$

Sfruttando quella che viene definita nel seguito penalty mista, si possono aggregare gli indici psl calcolati e ottenere un livello di servizio aggregato capace di descrivere, da un lato, la situazione in cui il livello di servizio è uguale o superiore al valore stabilito dal provider, dall'altro, di penalizzare l'indice quando questo non raggiunge il valore critico scelto.

Il grafico in Figura 3.6 mette a confronto le funzioni di penalty e i rispettivi indici di livello di servizio percepito. Nell'esempio, per la penalty mista, si è scelto un valore critico indicativo pari a $sl^* = 0.7$.

Una volta sviluppato questo metodo per l'aggregazione dei dati è possibile realizzare la funzione *getHostForVm* dell'algoritmo del *Capacity Planner* illustrata nella Sezione 3.2 (linea 8 dell'Algoritmo 1).

Riassumendo il procedimento complessivo, dopo aver stimato la *demand* (Sezione 3.3) delle singole macchine virtuali, già allocate nel datacenter, si valuta la possibilità di aggiungere iterativamente nuove macchine virtuali. La *demand* di queste ultime viene considerata sempre pari al 100%.

Tramite la funzione *getHostForVm*, l'algoritmo è in grado di stabilire quale host sia in grado di rispettare il livello di servizio critico garantito. Questo avviene calcolando il livello di servizio aggregato, sfruttando la penalty mista.

In sintesi, specificato il livello critico di livello di servizio che si vuole garantire, la funzione controllerà che $asl \geq sl^*$. Se la condizione è rispettata l'host verrà considerato in grado di allocare la macchina virtuale selezionata e sarà restituito come parametro dalla funzione *getHostForVm*. In caso negativo si procederà alla selezione di nuovo host e al ricalcolo del servizio di livello aggregato.

Nel caso in cui nessun host sia in grado di ospitare la macchina virtuale, si procederà alla creazione di nuovo host da inserire nell'architettura del datacenter.

Al fine dell'algoritmo si avrà a disposizione una possibile configurazione dell'infrastruttura del datacenter, in grado di supportare i carichi di lavoro stimati.

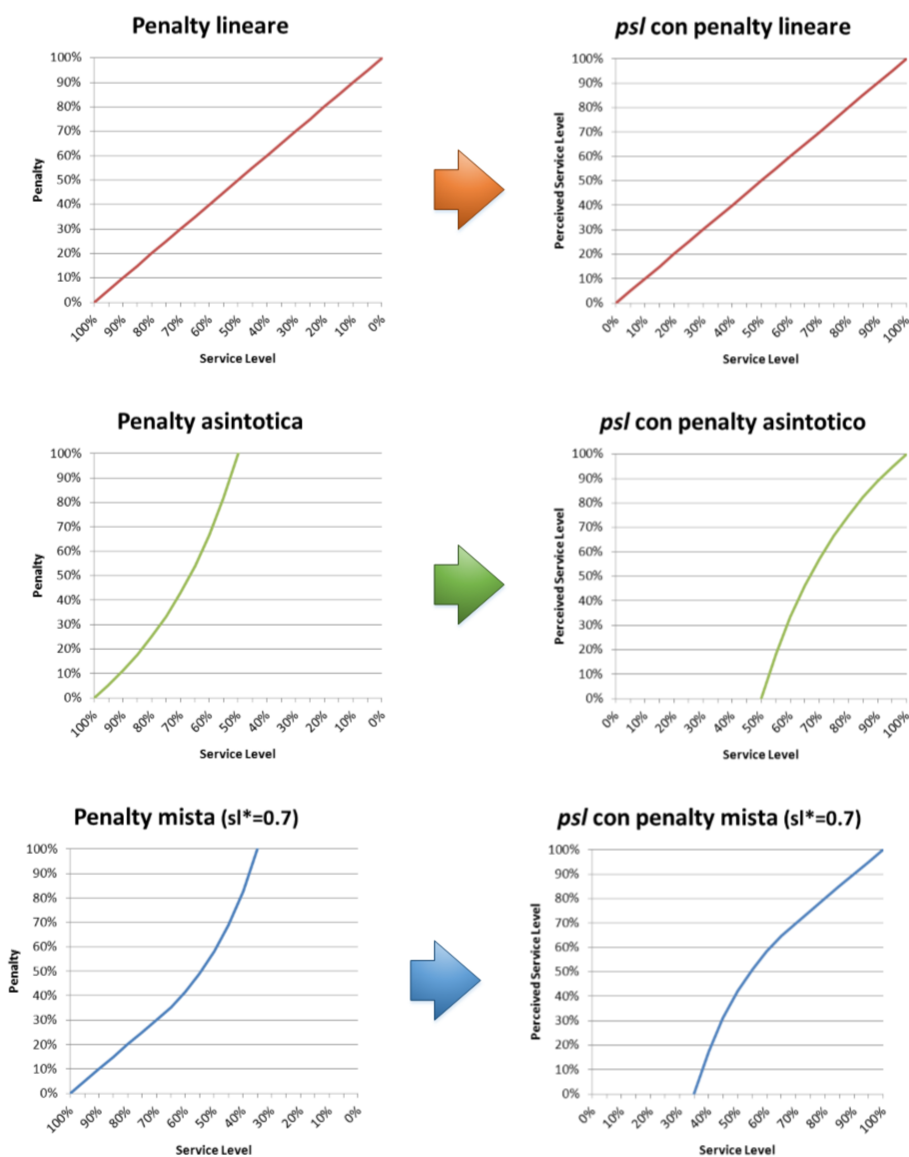


Figura 3.6: Livello di servizio percepito (psl) ottenuto applicando diverse funzioni di penalità

Capitolo 4

Metodi per l'allocazione dinamica di macchine virtuali

La soluzione proposta con il *CPA* illustrata nel Capitolo 3, pur ben adattandosi ad uno scenario di un datacenter di larga scala, evidenzia alcuni punti critici nell'analisi specifica di un host singolo. Più nel dettaglio, l'impossibilità della predizione dell'utilizzo di macchine virtuali non ancora allocate, genera un gap di conoscenza che può portare ad un sovradimensionamento del datacenter e che si riduce solo in periodi di tempo successivi. Infatti, dal momento in cui le macchine nuove cominciano ad essere effettivamente allocate, il *CPA* riesce a stimare in maniera sempre più corretta il carico di lavoro distribuito nel datacenter e la quantità di eventuali nuovi host necessari.

Senza mettere in discussione la bontà del *CPA*, che risponde ad un'esigenza periodica del provider per il controllo del datacenter, in questo capitolo si vuole affrontare la seconda sfida di questo elaborato: la decisione dinamica di allocazione. Dato lo stato corrente di un host, si vuole stabilire, a runtime, se esso sia in grado di soddisfare una potenziale richiesta di allocazione da parte di una macchina virtuale.

La soluzione proposta fa uso non soltanto della *demand* delle macchine virtuali attualmente allocate e di informazioni statiche, quali il numero di $vCPU$ e di $pCPU$, ma anche della predizione della *demand* della nuova macchina. Il comportamento della nuova macchina virtuale, che nel *CPA* veniva considerato impredicibile, si assume, questa volta, simile al comportamento delle macchine già allocate.

Utilizzando un istogramma della *demand* (Sezione 3.3.1 del Capitolo 3), costruito con i dati di tutte le macchine virtuali allocate, si vogliono stimare gli indici sl^{RT} e sl^{T}, introdotti nel capitolo precedente, che si potrebbero

ottenere con l'allocazione della nuova macchina. In questo modo è possibile dare una prima valutazione alla richiesta di allocazione.

La modalità di utilizzo dell'istogramma della *demand* per il calcolo degli indici sl^{RT} e sl^{T} sono oggetto della Sezione 4.1 in cui viene esposto il metodo definito *standard*.

La Sezione 4.2 estende la soluzione del metodo *standard* in modo da garantire quello che viene definito livello di *Reservation*, o semplicemente *Reservation*. La *Resevation* è un indice più stringente sul servizio offerto, in quanto obbliga il provider a garantire sempre un livello di *Run* assoluto e non relazionato alla *demand*. In altre parole una *Reservation* del 50% assicura che la macchina virtuale riceverà un valore di $r^{\%} = d^{\%}$ finché $d^{\%} \leq 50\%$. L'introduzione del metodo con *Reservation* è utile per offrire un approccio meno aggressivo nell'allocazione di risorse.

Nella Sezione 4.3, invece, viene ulteriormente esteso l'approccio precedente, proponendo un metodo di analisi per host che offrono la tecnologia Just in Time Activation, o *JiTA*, implementata nell'architettura del sistema cloud oggetto dell'elaborato. Peculiarità del *JiTA* è la possibilità che viene data all'utente di configurare la macchina virtuale, sia lato hardware, sia lato applicativo, prima ancora che questa venga effettivamente allocata sull'host. Questo comporta l'aggiunta di alcune variabili che nei metodi precedenti non vengono prese in considerazione.

Dato lo scenario aziendale in cui è stato svolto il lavoro di questo elaborato, è stato importante impostare un'analisi che potesse evidenziare l'impatto di questo tecnologia. Una volta elaborato il modello per l'analisi del *JiTA*, infatti, è stato possibile effettuare simulazioni per valutare la sua effettiva influenza. Metodologie e risultati delle simulazioni vengono descritti nella Sezione 4.4.

4.1 Metodo standard

L'idea chiave del primo approccio è accettare la richiesta di allocazione solo se la *demand* attesa della nuova macchina virtuale sommata alla *demand* corrente permette di garantire a tutte le macchine virtuali un *SL* uguale o maggiore a quello pattuito tra cliente e provider.

Supponendo di non avere alcuna informazione riguardo lo scopo applicativo per cui vengono usate le macchine virtuali, è stato ritenuto ragionevole attendersi che la nuova macchina virtuale mantenga un comportamento statisticamente simile a quelle attualmente allocate.

Sotto questa ipotesi è possibile costruire un istogramma della *demand* che profili il comportamento della nuova macchina virtuale, utilizzando i valori di *Run* e *Wait* di tutte le macchine virtuali allocate.

Come nel *CPA*, anche in questo caso è necessario stimare la *demand* delle macchine virtuali già esistenti. Sfruttando il *Demand Estimator*, il cui algoritmo è illustrato nella Sezione 3.3 del Capitolo 3, è possibile generare un array con i valori della *demand*, istante per istante, delle macchine virtuali attualmente allocate.

Nota la *demand* totale viene valutato il *SL* a fronte di una serie di richieste di utilizzo da parte della nuova macchina virtuale secondo gli intervalli dell'istogramma elaborato. Ogni *sl* viene quindi moltiplicato per la percentuale dell'intervallo rappresentato e viene quindi sommato agli altri *sl*.

Questo procedimento viene riassunto dalla formula nell'Equazione 4.1 che fa uso di un istogramma I_K di K intervalli per calcolare $\bar{sl}^{RT}(t)$, indice di *Respose Time* atteso, in un certo istante t:

$$\bar{sl}^{RT}(t) = \frac{\sum_{k=1}^{K} sl^{RT}(\mathcal{D}_k^t, N_{pCPU}) \cdot i_k}{\sum_{k=1}^{K} i_k} \tag{4.1}$$

e $\bar{sl}^{T}(t)$, indice di *Throughput* atteso, in un certo istante t:

$$\bar{sl}^{T}(t) = \frac{\sum_{k=0}^{K} sl^{T}(\mathcal{D}_k^t, N_{pCPU}) \cdot i_k}{\sum_{k=1}^{K} i_k} \tag{4.2}$$

con

$\mathcal{D}_k^t$ insieme delle coppie *demand* e numero di *vCPU* di ogni macchina v all'istante di tempo t, in cui il valore della *demand* della nuova macchina virtuale è dato da $\frac{k}{K}$.

Il metodo *standard*, quindi, prevede quattro passaggi:

1. Stima della *demand* delle macchine virtuali secondo l'Algoritmo 3 del Capitolo 3 che genera un array *demand[]* da salvare nella struttura dati delle macchine virtuali dell'host (es. la stima della *demand* della v-esima macchina virtuale viene salvata in *host.vms[v].demand[]*).

2. Calcolo degli indici $\bar{sl}^{RT}$ e $\bar{sl}^{T}$ per ogni istante di tempo t. Gli indici vengono calcolati utilizzando gli array contenti la *demand* stimata delle

macchine virtuali già attive e un comportamento della nuova macchina virtuale profilato da l'istogramma $I_K^{N \cdot vms.lenght}$ di K intervalli, generato con gli array *demand[]* di lunghezza N di *vms.lenght* macchine esistenti nell'host.

3. Aggregazione degli indici attesi calcolati in tutti gli N istanti di tempo.

4. Valutazione della richiesta: se il valore degli indici aggregati è conforme al livello di servizio richiesto, la richiesta di allocazione viene ritenuta accettabile.

4.2 Metodo con livello di *Reservation*

Il metodo precedente ha una buona resa quando la stima della *demand* delle macchine virtuale è accurata e quando i valori stimati riflettono un comportamento valido anche nel periodo subito successivo la valutazione. Tuttavia il metodo standard può rivelarsi una strategia troppo aggressiva per la scelta di allocazione della macchina virtuale nel caso in cui il comportamento registrato delle macchine virtuali differisca molto da quello che esse avranno nell'immediato futuro. Di seguito viene illustrato un esempio dello scenario appena descritto.

Un sottoinsieme delle macchine virtuali allocate su un host è caratterizzato da macchine il cui scopo applicativo è quello di entrare in funzione in caso di *emergenza* (es. vengono allocate per assicurare un certo livello di fault-tollerance). Assumendo che il tasso di malfunzionamento (hardware o software) dell'host su cui è allocata la macchina che svolge la funzione applicativa sia abbastanza basso, è ragionevole supporre che la macchina virtuale di emergenza abbia un consumo di risorse molto basso (i.e. la macchina si trova in modalità *stand-by*). Fare una stima della *demand* della macchina virtuale di supporto dai dati ottenuti nel periodo di *stand-by*, darebbe una valutazione delle performance offerte dall'host ben lontana da quella riscontrabile nel caso in cui la macchina virtuale di supporto uscisse dalla modalità di *stand-by*. Con l'uso del metodo standard l'algoritmo valuterebbe correttamente l'impossibilità di allocazione solo dopo l'entrata in funzione della macchina d'emergenza.

In questo scenario l'host potrebbe accettare un numero troppo elevato di macchine virtuali e, non appena una macchina di supporto cominciasse a richiedere l'utilizzo di risorse, l'host andrebbe subito in stato di overcommit. Il degrado delle performance risulterebbe con buone probabilità non accettabile. Per impedire questa situazione viene introdotto il concetto di livello di *Reservation* o *RL* (Reservation Level).

Il *RL* garantisce un servizio minimo assoluto (espresso sempre in percentuale), non relazionato con la *demand* della macchina virtuale. In altre parole dato un certo valore $0 < rl^{\%} < 1$, ogni macchina virtuale ha la certezza che nel caso di un picco di *demand* contemporaneo da parte di tutte le macchine virtuali $r^{\%} \geq rl^{\%}$.

Per assicurare il *RL*, il metodo con livello di *Reservation*, dato un parametro $rl^{\%}$, garantisce ad un insieme $\mathcal{VM}_h$ di macchine allocate su un host h, oltre al livello di servizio, la seguente relazione:

$$\frac{N^{pCPU}}{\sum\limits_{u \in \mathcal{VM}_h} N_u^{vCPU}} \geq rl^{\%} \tag{4.3}$$

Il processo di decisione sulla richiesta di allocazione, quindi, consta di due fasi:

1. Valutazione del *RL*: viene valutata la relazione sopra citata. Se essa risulta valida il processo continua, altrimenti termina.

2. Valutazione secondo il metodo standard: vengono eseguite le quattro fasi del metodo standard per assicurare il *SL*. Se anche il *SL* viene rispettato la richiesta di allocazione viene considerata accettabile.

4.3 Just in Time Activation

L'esigenza da parte dei provider di minimizzare l'utilizzo delle risorse e quella da parte dei clienti di utilizzare l'ambiente virtuale solo in caso di reale necessità, ha portato HP a introdurre nella soluzione cloud una nuova funzionalità definita *Just in Time Activation*, o semplicemente *JiTA*.

Nel normale processo di acquisto la macchina virtuale viene allocata sull'host non appena la transazione viene considerata valida. Nel momento in cui la macchina virtuale è disponibile, l'utente può configurare la stessa in previsione di un utilizzo che può non essere immediato. L'intervallo che intercorre tra l'acquisto e l'utilizzo in produzione della macchina virtuale si traduce in spreco di risorse da parte dell'host e in costi di licenza da parte dell'utente.

L'idea del *JiTA* nasce con l'intento di posticipare l'allocazione della macchina virtuale all'instante in cui l'utente ha l'effettiva intenzione di utilizzarla. In questo scenario anche la fase di configurazione della macchina virtuale (es. sistema operativo, applicativi, etc.) viene considerata una fase automatizzabile da un processo che si avvia alla richiesta di allocazione. Questo

processo migliora l'utilizzo delle risorse (i.e. CPU, bandwith, ma anche licenze software) consentendo un risparmio al cloud provider che si traduce in offerte più convenienti anche per i clienti.

Si vuole in questa sezione proporre un modello che riesca a sfruttare questa caratteristica nel momento della richiesta di acquisto di una macchina virtuale nel sistema.

A differenza delle soluzione proposte nelle Sezioni 4.1 e 4.2, è necessario considerare il processo di utilizzo della macchina virtuale suddiviso in due stati:

Off rappresenta lo stato iniziale in cui si trova la macchina virtuale appena termina la fase di acquisto;

On rappresenta lo stato di un macchina virtuale che viene allocata nell'host e che, quindi, consuma risorse.

A questi due stati vengono aggiunti gli stati di:

Birth rappresenta lo stato di una macchina virtuale durante la fase di acquisto;

Death rappresenta lo stato di un macchina virtuale che viene rimossa dal sistema e quindi dall'host.

Nota, quando la macchina virtuale si trova negli stati di *Birth* e *Death* il sistema non ha alcuna conoscenza sull'esistenza della stessa. La Figura 4.1 illustra il grafico del modello a stati che rappresenta la fruizione del servizio IaaS di tipo *JiTA*.

Inizialmente la macchina virtuale si trova in stato di *Birth*, è quindi in attesa che l'utente l'acquisti e che il sistema confermi la transazione. Appena la transizione viene accettata la macchina virtuale entra in stato di *Off* e viene considerata dal sistema come un'istanza che potenzialmente potrebbe richiede risorse.

Dopo un certo tempo medio t_{off}, il sistema riceve la richiesta di allocazione della macchina virtuale. Essa entra in stato di *On* e comincia effettivamente a consumare risorse. Una volta terminato l'utilizzo, dopo un intervallo t_{on}, l'utente fa richiesta di rimozione della macchina virtuale che entra in stato di *Death* e non viene più conteggiata tra le macchine virtuali del sistema.

Alla luce di questo, quello che nelle sezioni precedenti veniva considerata richiesta di *allocazione*, in questo scenario diventa una semplice richiesta di *associazione*, associazione della macchina virtuale ad un host.

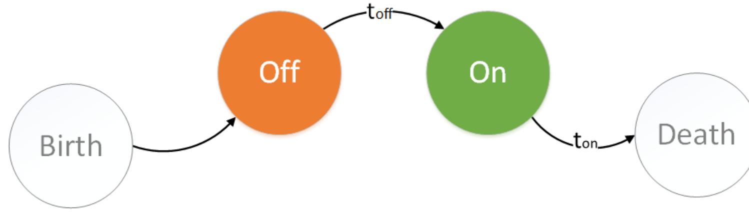


Figura 4.1: Modello a stati dell'utilizzo di una macchina virtuale

Per poter valutare la richiesta di associazione diventa indispensabile conoscere altre due misure:

tempo di off tempo in cui la macchina virtuale rimane in stato di *Off*;

tempo di on tempo in cui la macchina virtuale rimane in stato di *On*.

Nota una certa distribuzione per entrambi i tempi, è possibile raffinare il metodo standard per valutare la richiesta di associazione. Per far ciò è necessario definire alcune variabili:

$d_v^{\%}(t)$ *demand* della macchina v all'istante t

$\bar{d^{\%}}$ *demand* media delle macchine in stato di *On*

t_v^{on} tempo di permanenza delle macchina virtuale v nello stato *On*

$P_{T_{off}}$ funzione di ripartizione dei tempi di *Off* delle macchine virtuali del sistema

$P_{T_{on}}$ funzione di ripartizione dei tempi di *On* delle macchine virtuali del sistema

$\mathcal{VM}_{off}$ insieme delle macchine virtuali in stato di *Off*

$\mathcal{VM}_{on}$ insieme delle macchine virtuali in stato di *On*

Note le variabili è possibile calcolare

$\bar{d}_v^{\%}(t)$ *demand* attesa delle macchina virtuale v in stato di *On* all'istante t

come

$$\bar{d^{\%}}_v(t) = d_v^{\%}(t) \cdot \frac{(1 - P_{T_{on}}(X \leq t_v^{on} + t))}{(1 - P_{T_{on}}(X \leq t^{on}))} \tag{4.4}$$

e

$\bar{d}_v^{\%}(t)$ *demand* attesa delle macchina virtuale v in stato di *Off* all'istante t

come

$$\bar{d\%}_v(t) = \bar{d\%} \cdot \frac{P_{T_{off}}(X \le t_v^{off} + t) - P_{T_{off}}(X \le t_v^{off})}{1 - P_{T_{off}}(X \le t_v^{off})} \tag{4.5}$$

L'Equazione 4.4 sfrutta il teorema di Bayes secondo cui, dati due eventi A e B, è valida la relazione

$$P(A|B) = \frac{P(A) \cdot P(B|A)}{P(B)}$$

con $P(A)$ e $P(B)$, rispettivamente, la probabilità che si verifichi l'evento A e la probabilità che si verifichi l'evento B e con $P(A|B)$ e $P(B|A)$ la probabilità che si verifichi l'evento A dato B e viceversa.

Nell'Equazione 4.4 l'evento A è definito come *la macchina virtuale v è in stato di On per almeno un tempo $t^{on} + t$* e l'evento B come *la macchina virtuale v è in stato di On per almeno un tempo t^{on}.*

Nota $P_{T_{on}}(x)$ la funzione di ripartizione di t^{on} risulta che:

$$P(A) = P_{T_{on}}(X > t_v^{on} + t) = 1 - P_{T_{on}}(X \le t_v^{on} + t) \tag{4.6}$$

$$P(B) = P_{T_{on}}(X > t_v^{on}) = 1 - P_{T_{on}}(X \le t_v^{on}) \tag{4.7}$$

Sapendo che, per definizione, $t \ge 0$, si ha che l'evento $B|A$ (i.e. *la macchina virtuale è in stato di On per almeno un tempo t_v^{on}, sapendo che è in stato di On per almeno un tempo $t_v^{on} + t$*) è sempre verificato e cioè $P(B|A) = 1$.

Per l'Equazione 4.5, invece, si fa riferimento alla definizione di probabilità condizionata secondo cui:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

con A definito come *la macchina virtuale v è in stato di Off al massimo per un tempo $t^{off} + t$* e B come *la macchina virtuale v è in stato di Off per almeno un tempo t^{off}.*

Poiché le formule del calcolo del livello di servizio atteso (Equazioni 4.1 e 4.2) prevedono che la *demand* della nuova macchina sia pari a $\frac{k}{K}$, essa viene ridotta secondo la formula seguente:

$$\bar{d\%} = \frac{k}{K} \cdot P_{T_{off}}(X \le t) \tag{4.8}$$

Nota, nell'Equazione 4.8 non è necessaria l'applicazione della definizione di probabilità condizionata poiché l'istante in cui la macchina virtuale entra in stato di *Off* è considerato lo stesso istante dell'inizio dell'esecuzione del algoritmo di Dynamic Decision.

Come già veniva fatto nel metodo standard, anche in questo caso si potrebbe valutare il livello di servizio con le Equazioni 4.1 e 4.2 utilizzando la *demand* attesa al posto di quella reale (parallelamente anche la *demand* della macchina virtuale nuova verrebbe ridotto secondo l'Equazione 4.8). Questa soluzione, però, comporterebbe una valutazione del livello di servizio troppo ottimista. È evidente, infatti, che l'utilizzo delle Equazioni 4.4, 4.5 e 4.8, in accordo con il modello probabilistico, al crescere di t, porterebbe ad un graduale ridimensionamento del valore della *demand*. Di contro, però, il metodo standard non considera la possibilità che alcune macchine virtuali potrebbero, ad un certo istante di tempo, non richiedere più risorse.

Un compromesso che è stato ritenuto accettabile è stato quello di valutare $\bar{sl}_v(t)$, livello di servizio atteso per ogni macchina virtuale v considerando la *demand* della macchina v pari a $d^{\%}(t)$ e, per le altre associate all'host, pari a $\bar{d}^{\%}(t)$.

A questo è stata aggiunta una ulteriore correzione che considera la *demand* della macchina v pari $d^{\%}(t)$ anche nel caso in cui $t^{on} > \mu_{P_{Ton}} + \sigma_{P_{Ton}}$ con

$\mu_{P_{Ton}}$ media della funzione di ripartizione $P_{T^{on}}$;

$\sigma_{P_{Ton}}$ deviazione standard della funzione di ripartizione $P_{T^{on}}$.

Questo accorgimento permette di considerare le macchine virtuali che presentano un lungo periodo di *On* come sempre accese, condizione che appare ragionevole se viene superato un tempo maggiore della media sommata alla deviazione standard.

Al metodo standard viene quindi affiancato il metodo *JiTA* che si compone di quattro passaggi:

1. Stima della *demand* delle macchine virtuali secondo l'Algoritmo 3 del Capitolo 3 che genera un array *demand[]* da salvare nella struttura dati delle macchine virtuali dell'host.

2. Calcolo degli indici attesi $\bar{sl}_v^{RT}$ e $\bar{sl}_v^{T}$ per ogni istante di tempo t *per ogni macchina virtuale* v secondo $P_{T_{off}}$ e $P_{T_{on}}$. Gli indici attesi vengono calcolati utilizzando gli array contenti la *demand* stimata delle macchine virtuali già attive e un comportamento della nuova macchina virtuale profilato da l'istogramma $I_K^{N \cdot vms.lenght}$ di K intervalli, generato con gli array *demand[]* di lunghezza N di *vms.lenght* macchine esistenti nell'host.

3. Aggregazione degli indici attesi calcolati in tutti gli N istanti di tempo di lunghezza T_{sample}.

4. Valutazione della richiesta di *associazione*: se il valore degli indici aggregati è conforme al livello di servizio richiesto, la richiesta di associazione viene ritenuta accettabile.

4.4 Simulazioni di verifica

L'implementazione dell'Algoritmo di Decisione Dinamica, o più semplicemente *DDA* (*Dynamic Decision Algorithm*), ha permesso la realizzazione di simulazioni di verifica. L'obiettivo delle simulazioni è stato quello di misurare la validità degli approcci mostrati tramite la comparazione dei risultati delle analisi e i risultati ottenuti dalle simulazioni sul sistema reale.

Il secondo confronto, necessario a validare la soluzione proposta, è quello che viene fatto sul livello di servizio aggregato. Nel momento in cui il *DDA* conferma la possibilità di aggiungere una certa macchina virtuale, si vuole essere certi che il livello di servizio garantito venga rispettato durante la simulazione.

Questa sezione ripercorre i metodi introdotti per la realizzazione del *DDA*, strutturandosi anch'essa in tre parti:

- la Sottosezione 4.4.2 descrive e analizza le simulazioni effettuate per validare il *DDA* implementato secondo il metodo standard;
- la Sottosezione 4.4.3 analizza gli scenari ottenuti dalle precedenti simulazioni applicando, questa volta, il metodo con livello di *Reservation*;
- la Sottosezione 4.4.4, utilizzando i dati delle prime simulazioni, mette in luce in termini quantitativi quelli che potrebbero essere i vantaggi derivati dalla tecnologia *JiTA*.

Tutti i test di verifica sono stati realizzati utilizzando l'host descritto nella Sezione 2.1. Utilizzando una versione modificata dello script `CPU load`, invece, si è riusciti ad imporre alle macchine virtuali allocate un certo profilo di *demand*, impostando una *demand* dinamica.

4.4.1 Metodo di simulazione

Per la simulazione nell'ambiente di test si è proceduto nel seguente modo. Sono stati prelevati tre log di tre macchine virtuali quad-core presenti nel sistema cloud di produzione di HP. Ogni log riportava i dati di tempo di run, tempo di blocco e tempo di attesa con $t^{sample} = 20s$ e periodo di osservazione $T = 3600s$, per un totale di $N = 180$ campionamenti.

Ai log delle tre macchine virtuali, nel seguito denominate $vm1, vm2, vm3$, sono state aggiunte le misure delle macchine virtuali già esistenti nell'host (cfr. Sezione 2.1.1).

Quindi si è studiata la possibilità di aggiungere altre macchine virtuali in un ambiente che presentava un totale di 9 macchine virtuali:

- 1 macchina virtuale single-core
- 5 macchine virtuali dual-core
- 3 macchine virtuali quad-core

L'aggiunta delle tre macchine quad-core nasce dalla necessità di portare l'host di test in una situazione di utilizzo critica.

Fissati valori accettabili per il livello di servizio, sia per l'indice di *Response Time*, sia per l'indice di *Throughtput*, è stato utilizzato il *DDA* per stimare l'andamento del livello di servizio nel tempo.

Per poter ottenere dei dati sul livello del servizio reale è stato simulato l'utilizzo delle tre macchine quad-core allocandole sull'host ed eseguendo lo script `CPU load` modificato, utilizzando come input la *demand* stimata dai log.

Per campionare un possibile utilizzo della nuova macchina virtuale si è deciso di sfruttare l'istogramma della *demand*, realizzato con i log delle macchine considerate già allocate nell'host. Si è, quindi, deciso di estrarre in maniera casuale N valori, secondo la lunghezza dei log, in base alla distribuzione rappresentata dall'istogramma.

Per poter ottenere un campionamento più affidabile si è scelto di utilizzare un istogramma con $K = 10$, quindi, con larghezza degli intervalli pari al 10%.

Nota, i risultati mostrano i dati relativi alle tre macchine virtuali quad-core e alle macchine che si vogliono aggiungere nell'host. Infatti, poiché le sei macchine virtuali già allocate nell'host hanno un utilizzo decisamente limitato, presentano un livello di servizio poco significativo.

4.4.2 Simulazione con metodo standard

Test 1 Il primo test di verifica ha visto l'utilizzo del *DDA* per valutare l'aggiunta di una macchina virtuale quad-core. Come valori di livello di servizio critico si sono scelti $sl^{RT*} = 55\%$ e $sl^{T*} = 95\%$. La scelta di questi due valori critici determina, di fatto, due condizioni particolari:

1. l'indice di *Response Time* ha come livello critico del sistema il punto in cui il tempo di blocco è poco al di sotto del tempo di run (i.e.

se lo stesso lavoro venisse richiesto in un sistema non virtualizzato durerebbe circa la metà del tempo);

2. l'indice di *Throughput* critico molto alto garantisce l'esecuzione del carico di lavoro delle macchine virtuali.

Il *DDA* ha confermato l'impossibilità di aggiungere la macchina virtuale stimando i valori aggregati, con penalty mista, della macchina virtuale $vm3$ pari a $\widehat{asl}^{RT} = 53\%$ e $\widehat{asl}^{T} = 93\%$. Il livello di servizio aggregato dell'indice di *Throughput* è al di sotto del 95%. Lo stesso avviene per l'indice di *Response Time* che risulta essere minore del 55%, pertanto l'aggiunta di una macchina quad-core potrebbe comportare un degrado di performance non accettabile.

Di seguito vengono analizzati i risultati del test prendendo in esame prima i dati relativi all'indice di *Reponse Time*, quindi quelli relativi all'indice di *Throughput*.

Response Time Considerando che ogni indice di *Response Time* stimato, $\widehat{sl}^{RT}(n)$, risulta essere, dalla teoria, il limite inferiore per il livello di servizio reale, diventa ragionevole confrontare l'andamento $\widehat{sl}^{RT}(n)$ con il minimo tra gli indici di *Response Time* di tutte le macchine virtuali. Più formalmente siano, per esempio, $vm1, vm2, vm3$ le macchine allocate nell'host, l'indice $\widehat{sl}^{RT}(n)$ viene confrontato con $min(sl^{RT}_{vm1}(n), sl^{RT}_{vm2}(n), sl^{RT}_{vm3}(n))$. In questo modo si riesce a confrontare il profilo generico previsto dal *DDA*, con il profilo fittizio che considera il livello di servizio minimo tra le varie macchine virtuali in ogni istante.

Applicando questo criterio ai dati del *Test 1*, si ottiene il grafico in Figura 4.2. La linea blu rappresenta l'indice di *Response Time* minimo reale, mentre la linea rossa rappresenta l'indice stimato.

Come da previsione l'andamento dell'indice di *Response Time* reale è sopra la sua stima. Ciò non toglie che, pur non garantendo un'elevata accuratezza, la stima dell'indice riesce a modellare in maniera soddisfacente il suo andamento (l'errore medio tra stima e misura è pari a $\epsilon_{sl^{RT}} = 13.46\%$), evidenziando i punti in cui il carico di richiesta può essere critico.

La differenza tra valore reale e stima si ripercuote inevitabilmente anche sull'indice di livello di servizio aggregato. Come si può vedere dal grafico in Figura 4.3, infatti, a fronte di una previsione poco superiore al 50%, l'indice aggregato reale ottiene un valore al di sopra del 60%. Questo è di nuovo riconducibile al fatto che la stima del livello del tempo di blocco sia fatta nel

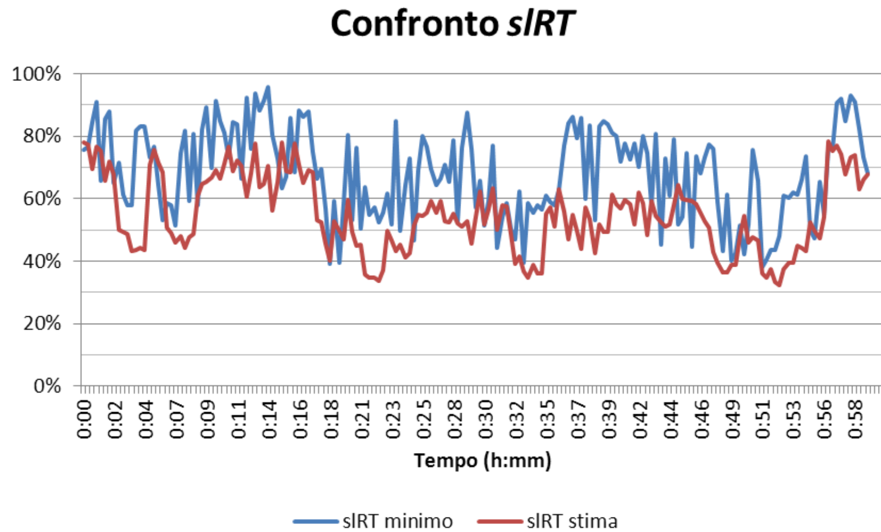


Figura 4.2: Confronto del livello di servizio di *Response Time* minimo stimato e misurato con l'aggiunta di una macchina quad-core ($newVm1$)

caso peggiore, quando, cioè, tutte le macchine virtuali hanno una *demand* uguale.

Throughput Se per l'indice sl^{RT} è stato opportuno valutare in maniera aggregata i dati di tutte le macchine virtuali, per l'indice sl^{T} si rivela più indicato analizzare i grafici di ogni singola macchina virtuale. Questo è possibile anche perché il *DDA* è in grado, in questo caso, di generare un grafico specifico per ogni macchina.

È importante notare che, essendo l'approccio illustrato nella Sezione 4.1 un metodo statistico, per la macchina nuova da allocare vi sia l'impossibilità di prevedere i picchi di richiesta. Ciò si ripercuote, inevitabilmente, nel confronto dell'indice di livello di servizio istante per istante. Di fatto per la nuova macchina virtuale, è importante constatare una correlazione soprattutto nell'indice aggregato.

Quello che si evince dal confronto tra i dati reali e la loro stima è un effettiva corrispondenza negli istanti in cui si verifica un calo di performance (evidenziati anche da un tempo di attesa vicino a $0s$).

Le Figure 4.4 e 4.5 mostrano i grafici delle macchine virtuali $vm1, vm2, vm3$ e $newVm1$. Mentre nei primi tre grafici si riesce a vedere una correlazione tra la stima e le misure dell'indice di *Throughput*, il grafico della nuova macchina virtuale $newVm1$, come già anticipato, ha un comportamento dell'indice reale che si discosta dalla stima.

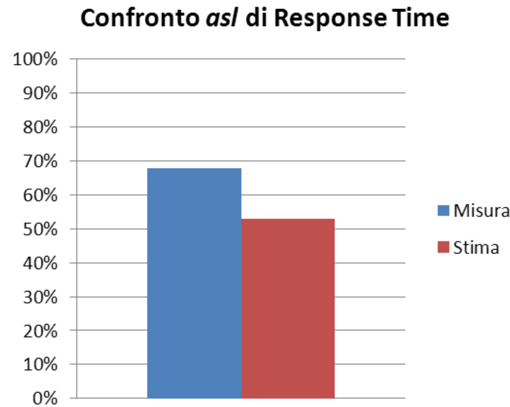


Figura 4.3: Confronto del livello di servizio di *Response Time* aggregato, asl^{RT}, con l'aggiunta di una macchina dual-core

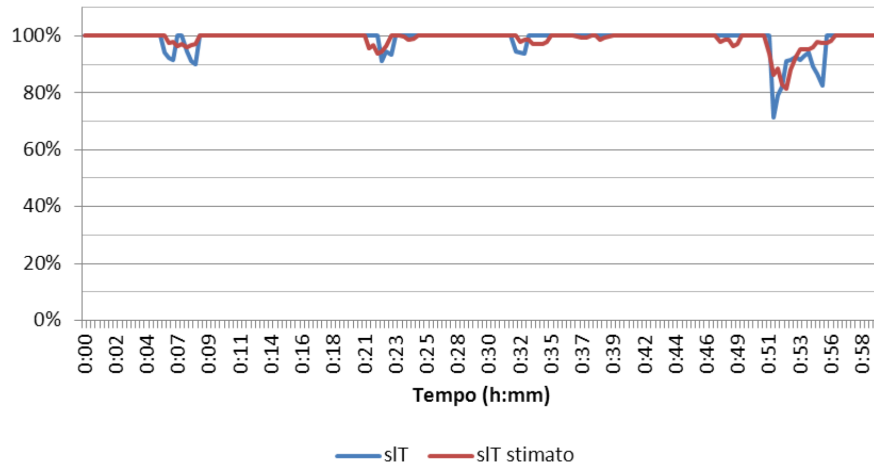
Il grafico in Figura 4.6 mostra l'asl^T delle macchine virtuali $vm1, vm2, vm3$ e $newVm1$. Si può osservare come, in questo caso, il valore previsto si avvicini molto di più al valore reale, pur sovrastimandolo leggermente. Questo deriva, probabilmente, dall'impossibilità dell'host di garantire il completo utilizzo in caso di situazioni molto vicine allo stato over-commit. In altre parole, il *DDA* trascura un certo tasso di overhead, per esempio, dovuto all'hypervisor che gestisce l'allocazione delle risorse alle macchine virtuale.

Ciononostante è importante osservare che il *DDA*, consigliava la non allocazione della nuova macchina virtuale a causa di un calo delle performance, sotto il livello critico ($sl^{T*} = 95\%$ e $sl^{RT*} = 55\%$). La simulazione ha confermato la previsione dell'algoritmo.

Test 2 Nella seconda simulazione, data la configurazione iniziale, si è valutata la possibilità di aggiungere una macchina virtuale dual-core. Gli indici di livello di servizio critico sono rimasti invariati rispetto al *Test 1* ($sl^{RT*} = 55\%$, $sl^{T*} = 95\%$).

Questa volta il *DDA* ha confermato la possibilità di aggiungere la macchina virtuale che, avendo 2 *vCPU* in meno della precedente, presenta un profilo di *demand* più compatibile con l'attuale configurazione.

Response Time Il grafico in Figura 4.7, che rappresenta il livello di servizio di *Response Time* registrato minimo e la sua stima, conferma nuovamente come i valori previsti dal *DDA* siano i limiti inferiori dell'indice misurato con analoga *demand* media.



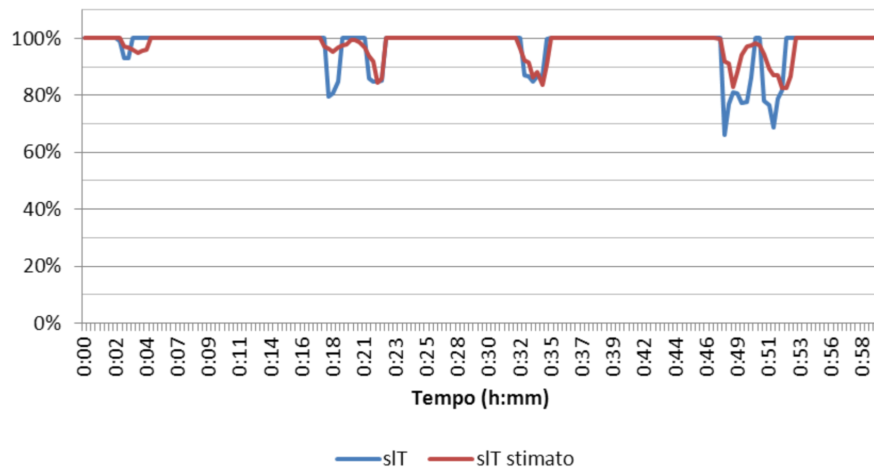
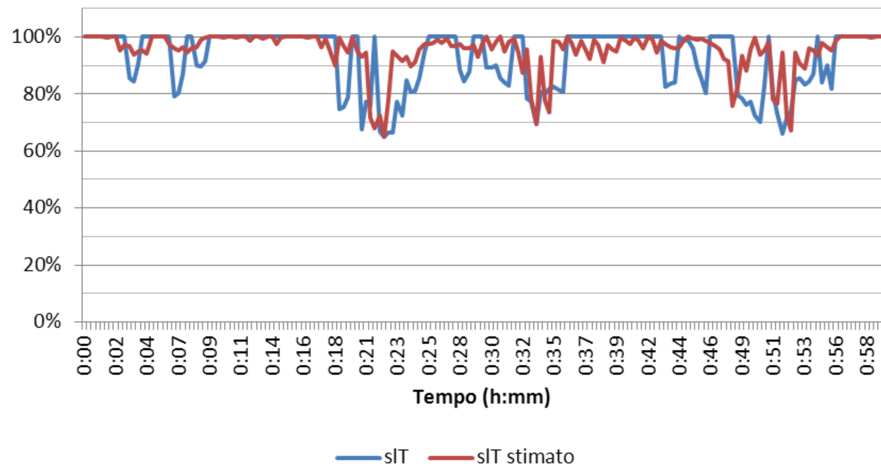


Figura 4.4: Confronto del livello di servizio di *Throughput* stimato e misurato per $vm1$ e $vm2$ con l'aggiunta di una macchina quad-core ($newVm1$)



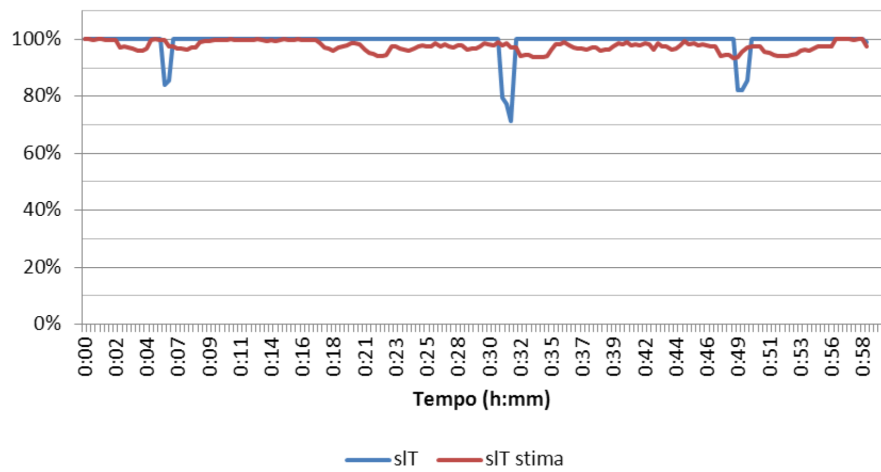


Figura 4.5: Confronto del livello di servizio di *Throughput* stimato e misurato per $vm3$ e $newVm1$ con l'aggiunta di una macchina quad-core ($newVm1$)

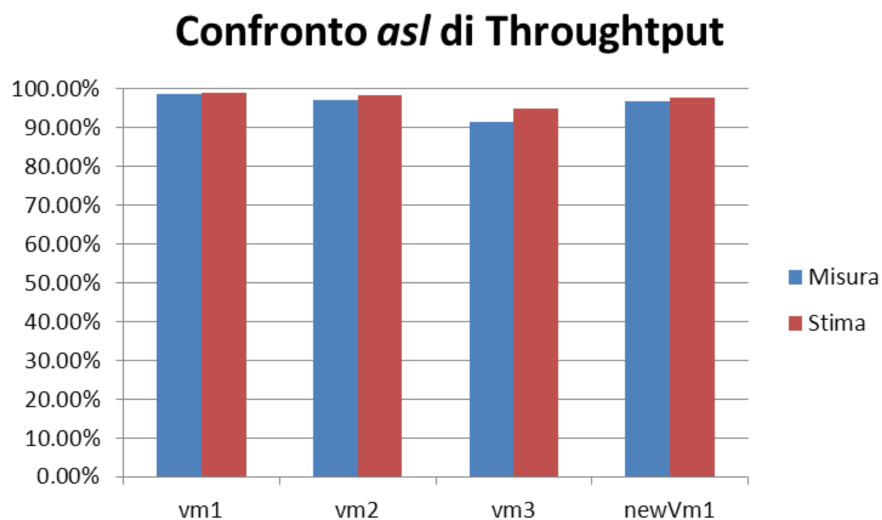


Figura 4.6: Confronto del livello di servizio di *Throughput* aggregato, asl^T, con l'aggiunta di una macchina quad-core ($newVm1$)

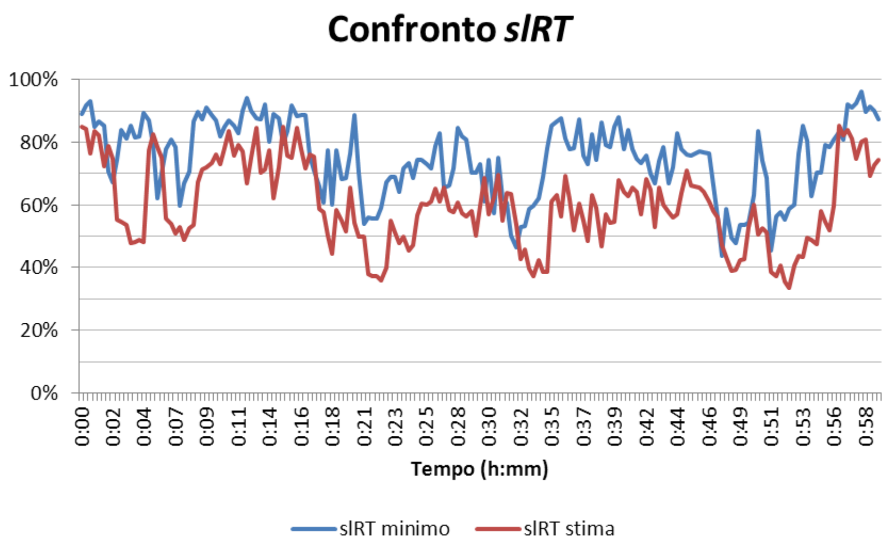


Figura 4.7: Confronto del livello di servizio di *Response Time* minimo stimato e misurato con l'aggiunta di una macchina dual-core ($newVm2$)

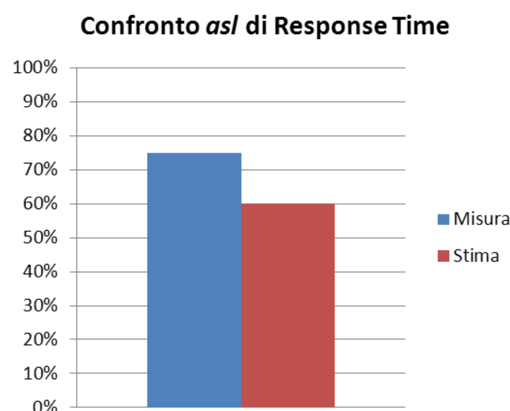


Figura 4.8: Confronto del livello di servizio di *Response Time* aggregato, asl^{RT}, con l'aggiunta di una macchina dual-core ($newVm2$)

È quindi immediato capire che anche il livello di servizio aggregato stimato risulti, ancora una volta, inferiore a quello reale. In particolare, come mostra il grafico in Figura 4.8, si ha che $\widehat{asl}^{RT} = 59\%$ e $asl^{RT} = 75\%$.

Throughput Se l'indice di *Response Time* denota alcuni cali di performance, non è lo stesso per l'indice di *Throughput* che tocca più volte il 100%. Essendo, in questo caso, i grafici delle singole macchine virtuali poco significativi si riportano solo i risultati del livello di servizio di *Throughput* aggregato, asl^{T}.

In Figura 4.9 si può notare come, per tutte le macchina virtuali, l'indice aggregato, sempre con penalty mista, confermi la possibilità effettiva di aggiungere la macchina virtuale dual-core, ottenendo per tutte le 4 macchine virtuali un livello di servizio di *Throughput* aggregato sopra al 95%.

Test 3 Con la terza simulazione si è voluto testare la possibilità di aggiungere una macchina single-core dato lo scenario del *Test 2*. Quindi, sfruttando i log della simulazione precedente, il *DDA* ha calcolato gli indici di *Response Time* e *Throughput* per questa nuova configurazione.

Riassumendo la simulazione prevedeva:

- 5 macchine virtuali (1 single-core, 4 dual-core) già presenti nel sistema cloud di HP
- 3 macchine virtuali (quad-core) simulate dai log precedenti
- 1 macchina virtuale (dual-core) simulata con il log del *Test 2*

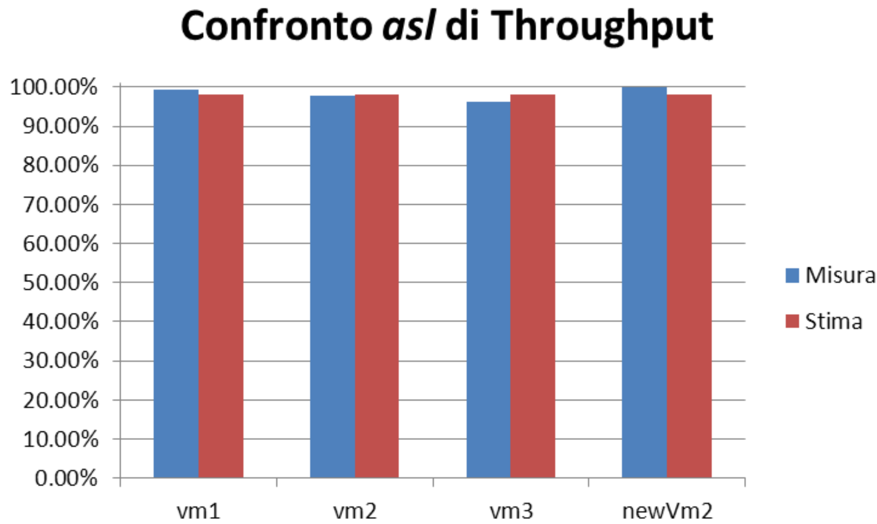


Figura 4.9: Confronto del livello di servizio di *Throughput* aggregato, asl^T, con l'aggiunta di una macchina dual-core ($newVm2$)

- 1 macchina virtuale (single-core) simulata con un log generato dal campionamento casuale fatto sull'istogramma della *demand* delle 9 macchine precedenti

Anche in questo contesto il *DDA* ha confermato la possibilità di aggiungere la nuova macchina virtuale, $newVm3$. Questo a conferma del fatto che lo scenario del *Test 1*, che rispetto a quello corrente ha 1 *vCPU* in più, sia un caso limite con performance poco al di sotto di quelle richieste.

Response Time Anche in questa situazione si è potuto registrare un gap tra il livello di servizio di *Response Time* stimato e misurato, questo ad ulteriore conferma della teoria. In Figura 4.10 è possibile vedere il grafico dell'andamento dell'indice di *Response Time* nel tempo. Anche in questo caso il livello di servizio aggregato (grafico in Figura 4.11) rispetta pienamente il livello di servizio critico, impostato sempre a $sl^{RT*} = 55\%$.

Throughput Per quanto riguarda l'indice di *Throughput* le osservazioni sono minime in quanto, fatta eccezione per la macchina virtuale $vm3$, che ha un carico di lavoro più elevato, le macchine virtuali mantengono il livello di servizio molto vicino al 100%. In Figura 4.12 viene confrontato l'indice aggregato delle cinque macchine virtuali simulate.

In conclusione si può affermare che il *DDA* con metodo standard risulti, nel complesso, un metodo valido per prevedere la possibilità di allocazione di

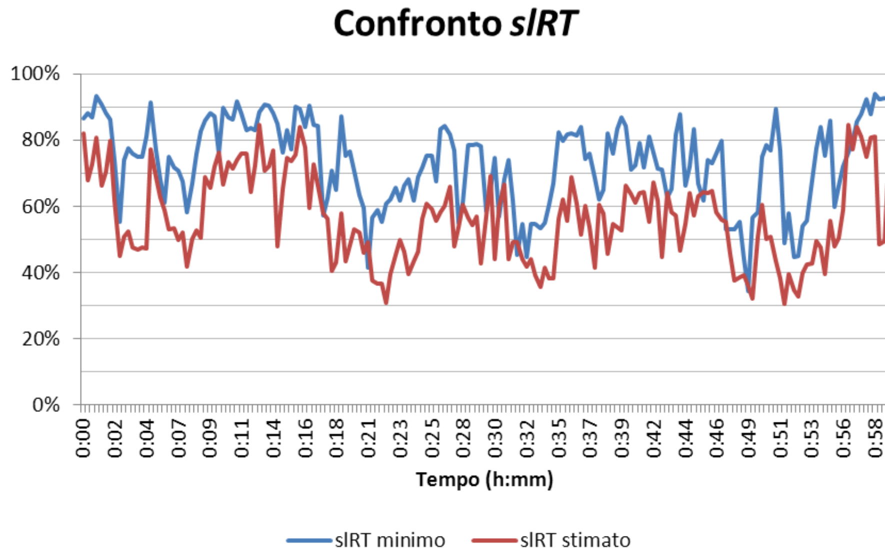


Figura 4.10: Confronto del livello di servizio di *Response Time* minimo stimato e misurato con l'aggiunta di una macchina single-core ($newVm3$)

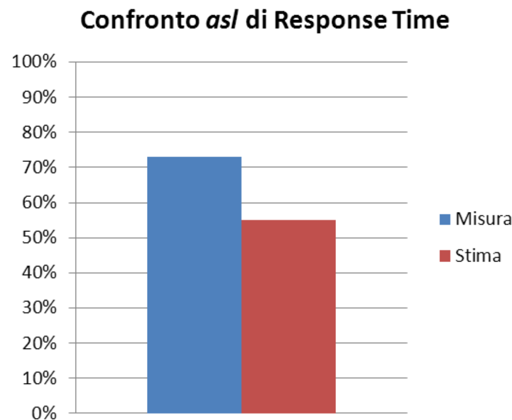


Figura 4.11: Confronto del livello di servizio di *Response Time* aggregato, asl^{RT}, con l'aggiunta di una macchina single-core

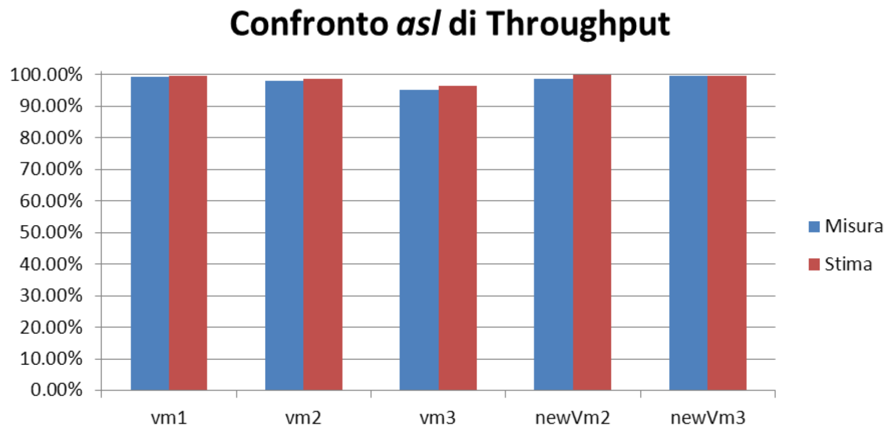


Figura 4.12: Confronto del livello di servizio di *Throughput* aggregato, asl^T, con l'aggiunta di una macchina single-core ($newVm3$)

una macchina virtuale. Questo è confermato dalle simulazioni, ovviamente, con tutte le limitazioni del caso:

- comportamento delle macchine virtuali già allocate molto simile a quello appena trascorso
- comportamento della nuova macchina virtuale in linea con la *demand* media delle altre macchine
- permanenza nel sistema di tutte le macchine virtuali attualmente allocate

4.4.3 Analisi scenario con livello di *Reservation*

Se nella teoria il *DDA* con *Reservation* può presentare alcune difficoltà, l'applicazione di questo metodo a monte del *DDA* standard è abbastanza immediato. Infatti, utilizzando gli scenari presentati nella Sezione 4.4.2, è possibile validare o meno l'inserimento di una nuova macchina virtuale previo il rispetto della relazione illustrata nell'Equazione 4.3.

Questa relazione presenta un vincolo più stringente poiché è indipendente dall'utilizzo delle risorse fisiche che fa ogni macchina virtuale. In particolare l'Equazione 4.3, impone, di fatto, un numero massimo di di *vCPU* che l'host può allocare.

Prendendo di nuovo in considerazione la configurazione illustrata nella Sezione 4.4.1 e volendo impostare una *Reservation* pari a $rl^{\%} = 35\%$, si ha che l'host ammette un massimo di 22 *vCPU*. Avendo la configurazione

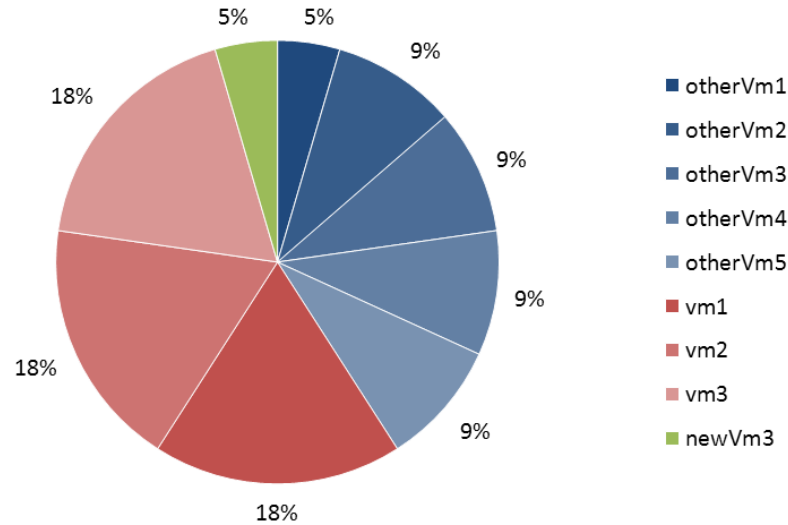


Figura 4.13: Ripartizione della capacità fisica della CPU in caso di picco simultaneo

iniziale un totale di 21 *vCPU* (1 macchina single-core, 4 macchine dual-core, 3 macchine quad-core), è possibile l'allocazione di una sola macchina virtuale single-core.

Poiché il *Test 3* prevedeva la possibilità di aggiungere all'host due macchine virtuali (1 dual-core, 1 single-core) e gli indici di *Throughput* e *Response Time* aggregati garantivano performance accettabili, è logico ritenere possibile anche l'aggiunta della sola macchina virtuale single-core.

Quindi, supponendo di aggiungere una macchina virtuale single-core alla configurazione iniziale, in caso di picco simultaneo di tutte le macchine virtuale, la risorsa fisica sarebbe divisa come nel grafico in Figura 4.13. In blu è indicata la quantità di CPU dedicata alle macchine attualmente allocate, in rosso la quantità di CPU dedicata alle macchine quad-core create per le simulazioni, in verde la quantità di CPU dedicata alla macchina nuova single-core. Di fatto, come mostra la Tabella 4.1, il tempo di run in percentuale di ogni macchina virtuale risulta $r^{\%} = 36\% \geq rl^{\%}$.

Nota, ricordando la definizione di $a^{\%}$, capacità di calcolo assegnata, (Sezione 1.4.1), il tempo di run di ogni macchina virtuale si ottiene con la formula

$$r^{\%} = a^{\%} \cdot \frac{N^{pCPU}}{N^{vCPU}}$$

A differenza del metodo standard, impostando un livello di *Reservation*

	# $vCPU$	% Host CPU ($a^{\%}$)	$R^{\%}$
otherVm1	1	5%	36%
otherVm2	2	9%	36%
otherVm3	2	9%	36%
otherVm4	2	9%	36%
otherVm5	2	9%	36%
vm1	4	18%	36%
vm2	4	18%	36%
vm3	4	18%	36%
newVm3	1	5%	36%

Tabella 4.1: Ripartizione della capacità fisica (% Host CPU) in caso di picco simultaneo

pari al 35%, si impedisce ad un potenziale provider di allocare sull'host una macchina virtuale con più di 1 $vCPU$. Quindi, se nello scenario visto nel *Test 2* la macchina virtuale dual-core veniva accettata, in questo caso essa non può essere accetta poiché la sua allocazione violerebbe la relazione dell'Equazione 4.3.

In definitiva, il *DDA* con livello di *Reservation*, nello scenario indicato, ammette l'allocazione di una sola macchina virtuale single-core.

4.4.4 Impatto della tecnologia *JiTA*

Misurare l'impatto della tecnologia *JiTA* all'interno di un sistema cloud è una sfida che non trova una soluzione banale. Nella Sezione 4.3 si è cercato di fornire un metodo che, espandendo il *DDA* standard, fosse in grado di fornire una stima degli indici aggregati delle macchine virtuali attualmente nel sistema e di quella che richiede l'allocazione.

Il metodo proposto, però, necessita di dati sui tempi medi di *On* e di *Off* che possono essere raccolti solo nel momento in cui il sistema entra realmente in produzione. Per comprendere quali potessero essere gli scenari vantaggiosi per questo tipo di tecnologia, si è operata una scelta per la modellazione dei tempi di *On* e di *Off*.

Intuitivamente per la distribuzione dei tempi di *On* si scelto di utilizzare una distribuzione Gamma ($\gamma(k, \theta)$) con $k \gg 1$. Ciò è motivato dal fatto che, ricordando che la distribuzione Gamma per $k = 1$ si riduce ad una esponenziale, la sua funzione di ripartizione, definita solo per valori positivi, mantiene una probabilità *molto bassa* per i valori vicino a 0. Quindi ha un punto di flesso, superato il quale la funzione tende più o meno rapidamente al 100%. In altre parole si può dire che la probabilità che la macchina virtuale

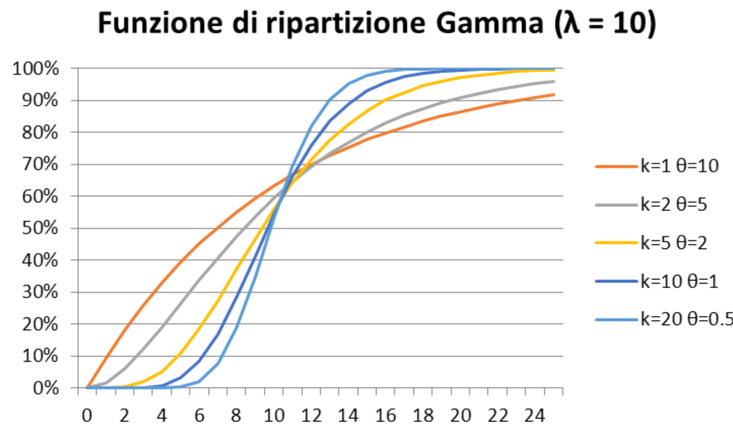


Figura 4.14: Esempi di funzioni di ripartizioni Gamma con media 10 e parametro k variabile

venga spenta poco dopo la sua accensione è molto bassa. Al contrario, superato un certo periodo di vita, il suo spegnimento diventa molto più probabile.

Analogamente anche per i tempi di *Off* si è scelta una distribuzione Gamma, questa con parametro k "sufficientemente" piccolo, con media inferiore alla distribuzione precedente. Questo fa sì che la distribuzione Gamma si avvicini maggiormente ad una distribuzione esponenziale, che modella, pertanto, un aumento della probabilità di accensione della macchina già nei primi istanti di vita, come ci si può aspettare da un comportamento reale della macchina virtuale.

La Figura 4.14 mostra alcune curve Gamma al variare del parametro k, mantenendo fissa a 10 la media della distribuzione.

Per poter, infine, analizzare alcuni scenari che verifichino l'impatto della tecnologia *JiTA*, è stato necessario dare un valore ai tempi di *On* delle macchine virtuali, così da applicare la formula dell'Equazione 4.4.

Scelti il valore k e la media delle due distribuzioni Gamma è stato utilizzato il *DDA* con l'implementazione del *JiTA* per stimare gli indici aggregati di *Response Time* e di *Throughput* al variare dei tempi di *On* di due macchine virtuali.

Impostando un valore medio pari a 90 giorni e $k = 20$ per il tempo di *On* ed un valore medio di 10 giorni e $k = 5$ per il tempo di di *Off*, si è potuto calcolare il parametro θ. Poiché il *DDA* calcola la probabilità di accensione e spegnimento secondo gli intervalli di campionamento del log, nel nostro caso pari a 20s, il valore medio di *On* corrisponde a $90 \cdot 24 \cdot 60 \cdot 3 = 388800$ istanti, quello di *Off* a $10 \cdot 24 \cdot 60 \cdot 3 = 43200$ istanti.

Noto che la media λ_γ di una distribuzione $\gamma(k, \theta)$ è pari a

$$\lambda_\gamma = k \cdot \theta$$

si ha che

$$\theta = \frac{\lambda_\gamma}{k}$$

con la cui equazione si può calcolare il valore θ per definire le distribuzioni di *On* e di *Off*.

Più precisamente per il tempo di *On* è stata utilizzata una distribuzione $\gamma(20, 19440)$, per il tempo di Off una distribuzione $\gamma(5, 8640)$.

Stabilite le due distribuzioni si è valutato l'indice aggregato di *Response Time* e l'indice aggregato di *Throughput* delle macchina *newVm1*, quad-core, facendo variare il tempo di *On* delle due macchine virtuali che richiedevano maggiori risorse: *vm1* e *vm3*.

Lo scenario della simulazione è il medesimo del *Test 1* illustrato nella Sezione 4.4.2 con un totale di 8 macchine virtuali già presenti nel sistema e 1 nuova macchina virtuale che richiede l'allocazione. Da notare il fatto che, essendo questa una simulazione per la valutazione preventiva della tecnologia, si è deciso di considerare uno scenario relativamente semplice in cui:

- non sono presenti nel sistema macchine virtuali in stato di *Off*;
- tutte le macchine virtuali presenti nel sistema, ad eccezione di *vm1* e *vm3*, hanno un numero di istanti in stato di *On* maggiore di $\mu_{\gamma(20,19440)} + \sigma_{\gamma(20,19440)}$

Oltre agli indici aggregati della macchina virtuale *newVm1*, per valutare i benefici che la tecnologia *JiTA* può apportare al sistema, si è tenuto conto della variazione degli indici delle due macchine virtuali *vm1* e *vm3*. Ricordando che il metodo proposto considera pari a 100% la probabilità che la macchina virtuale di cui si sta calcolando l'indice sia accesa, diventa interessante fare un confronto tra i dati ottenuti con l'analisi del metodo standard e quelli ottenuti utilizzando il metodo *JiTA*, che considera minore l'impatto immediato della nuova macchina virtual (*newVm1*).

I risultati delle simulazioni relativi alla macchina virtuale *newVm1* sono indicati nella Tabella 4.2 e nella Tabella 4.3, che indicano, rispettivamente, l'indice aggregato di *Response Time* e l'indice aggregato di *Throughput*. Le simulazioni sono state eseguite settando i tempi di *On* delle macchine virtuali *vm1* e *vm3* a 30, 60, 90, 120 giorni.

La crescita degli indici aggregati sulle righe delle tabelle è dovuto all'aumento dei giorni in cui la macchina virtuale *vm1* è in stato di *On*. La crescita

che viene evidenziata sulle colonne è dovuta, invece, all'aumento di giorni in stato di *On* della macchina *vm3*. Più il tempo di *On* si avvicina al tempo medio (90 giorni), più aumenta la probabilità che la macchina virtuale in oggetto si spenga.

Il metodo, però, tiene conto anche della possibilità che se una macchina virtuale supera un certo tempo di *On*, la probabilità di spegnimento si azzera. In altre parole, se una macchina virtuali è stata accesa per un numero di istanti maggiore di $\mu_{\gamma(20,19440)} + \sigma_{\gamma(20,19440)}$(Sezione 4.3), allora $t_{on} \to \infty$. Ciò è evidente rispettivamente nell'ultima riga e nell'ultima colonna delle Tabelle 4.2 e 4.3 dove gli indici aggregati si abbassano.

T^{on}_{vm3} \ T^{on}_{vm1}	**30**	**60**	**90**	**120**
30	52.11%	53.22%	63.46%	52.11%
60	54.06%	55.18%	65.41%	54.06%
90	71.41%	72.49%	81.55%	71.41%
120	51.11%	53.22%	63.46%	52.11%

Tabella 4.2: Valore dell'indice aggregato di *Response Time* di *newVm1* (4 *vCPU*) al variare del tempo di *On* di *vm1* e *vm3*

T^{on}_{vm3} \ T^{on}_{vm1}	**30**	**60**	**90**	**120**
30	97.64%	97.76%	99.03%	97.04%
60	97.69%	97.81%	99.08%	97.69%
90	99.17%	99.23%	99.76%	99.17%
120	97.64%	97.76%	99.03%	97.64%

Tabella 4.3: Valore dell'indice aggregato di *Throughput* di *newVm1* (4 *vCPU*) al variare del tempo di *On* di *vm1* e *vm3*

I grafici in Figura 4.15(a) e in Figura 4.15(b) mostrano i risultati graficamente. Come era naturale aspettarsi, quando entrambe le macchine virtuali raggiungono dei tempi di *On* vicini al valore medio (90 giorni), il loro impatto sul sistema diminuisce, consentendo alla macchina virtuale quad-core di migliorare l'asl^{RT} fino al 37% (scenario con tempo di *On* pari 30 giorni per *vm1* e *vm3* confrontato con scenario con tempo di *On* pari a 90 giorni per entrambe le macchine virtuali).

L'asl^{T}, essendo già vicino al 100%, ha un miglioramento più ridotto, ma comunque rilevante considerando che il valore critico è settato al 95%.

Confrontando gli scenari estremi ($t^{on} = 30$ e $t^{on} = 90$, espresso in giorni, per entrambe le macchine virtuali) si ha un miglioramento attorno al 1.6%.

Da notare che, con l'accortezza di considerare al 100% la *demand* della macchina virtuale di cui si calcolano gli indici, è possibile valutare il suo impatto sul sistema nel caso dovesse accendersi negli istanti subito successivi. In altre parole il *DDA* con il metodo JiTA calcola gli indici aggregati riducendo la *demand* delle macchine virtuali secondo il loro stato di *On* o di *Off*, ma mantenendo inalterata la *demand* della macchina di cui si vogliono conoscere gli indici.

Questo permette di avere una proiezione più reale delle performance che potrebbe misurare la nuova macchina virtuale nel caso dovesse accendersi in tempi molto *brevi*.

In queste condizioni è interessante valutare anche gli indici aggregati della macchina virtuale *vm1* e *vm3*. Questi valori forniscono un prima valutazione dell'impatto che può avere la tecnologia *JiTA* sull'host. Considerare gli indici aggregati delle macchine virtuali già presenti nell'host, infatti, permette di evidenziare come la performance misurate migliorino rispetto al metodo standard, poiché l'accensione della nuova macchina virtuale non è immediata.

I grafici in Figura 4.16 mostrano la variazione dell'indice di *Response Time* (Figura 4.16(a)) e di *Throughput* (Figura 4.16(b)) della macchina virtuale *vm1*, al variare del tempo di *On* della macchina virtuale *vm3*.

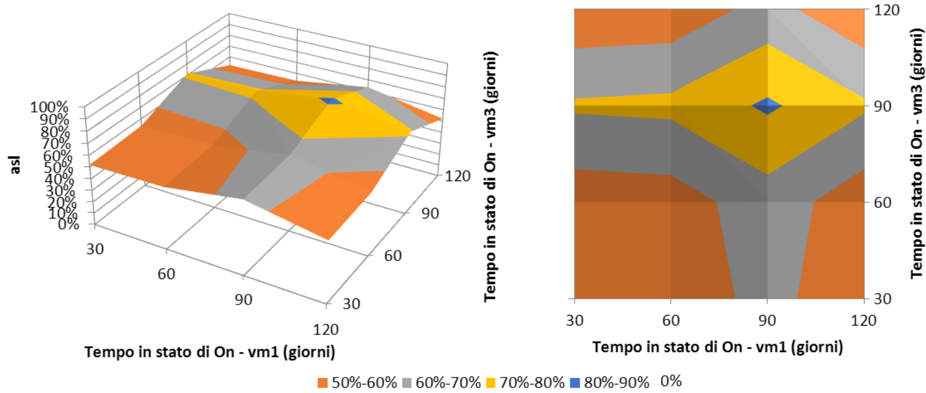
Analogamente vengono riportati i grafici in Figura 4.17 che mostrano la variazione dell'indice di Response Time (Figura 4.17(a)) e di Throughput (Figura 4.16(b)) della macchina virtuale *vm3*, al variare del tempo di *On* della macchina virtuale *vm1*.

In entrambi i grafici è presente anche l'incremento relativo rispetto al metodo standard calcolato sugli indici di *Respose Time* e *Throughput* ottenuti nel Test 1.

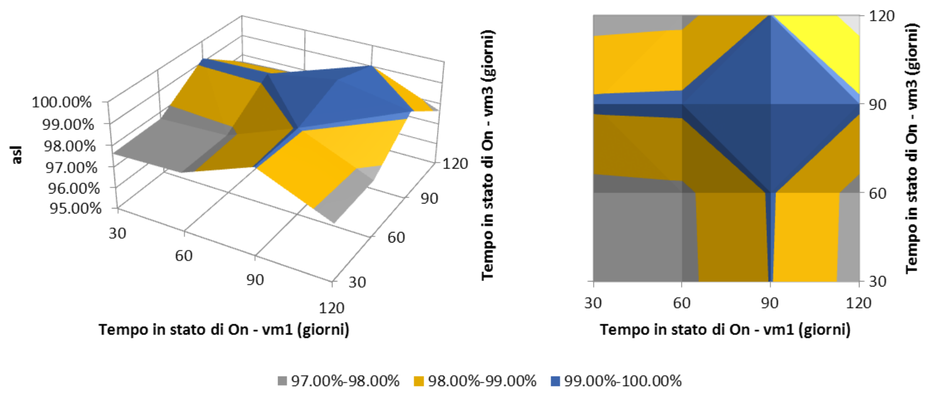
4.5 Conclusioni

I tre metodi di valutazione (metodo standard, metodo standard con livello di *Reservation* e metodo standard con *JiTA*) mostrano tre approcci per la soluzione della sfida iniziale: valutare la possibilità di allocazione di macchine virtuali con un numero di *vCPU* noto.

Riassumendo i risultati ottenuti in tutte le simulazioni, si può notare come il metodo standard si attesti come soluzione intermedia capace di permettere l'allocazione di due macchine virtuali per un totale di 3 *vCPU* oltre a quelle già presenti nel sistema.

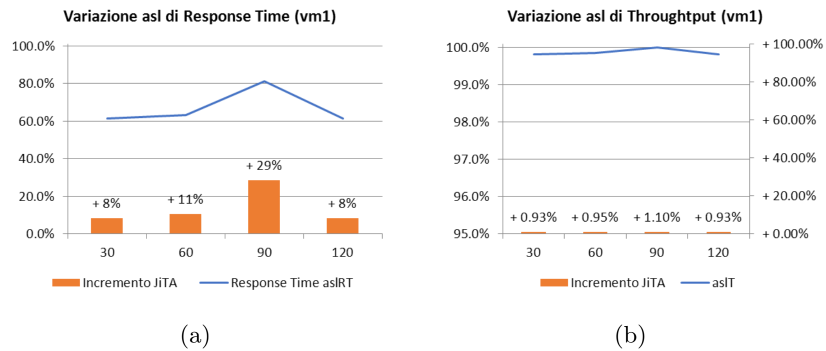


(a) Valore dell'indice aggregato di *Response Time* di *newVm1* (4 *vCPU*) al variare del tempo di *On* di *vm1* e *vm3*



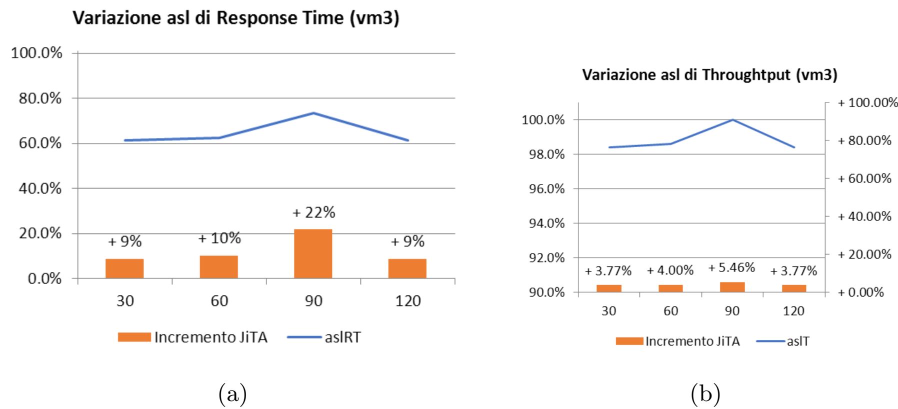
(b) Valore dell'indice aggregato di *Throughput* di *newVm1* (4 *vCPU*) al variare del tempo di *On* di *vm1* e *vm3*

Figura 4.15: Variazione degli indici aggregati di *newVm1*



(a) (b)

Figura 4.16: Variazione degli indici aggregati di *vm1* al variare del tempo di *On* di *vm3*



(a) (b)

Figura 4.17: Variazione degli indici aggregati di *vm3* al variare del tempo di *On* di *vm1*

Data la definizione di *Reservation*, il secondo metodo pone maggiori restrizioni sulle performance delle macchine virtuali adattandosi maggiormente a scenari in cui risulta difficile profilare la distribuzione della *demand* delle macchine virtuali. Questo metodo consentiva, nella simulazione, l'aggiunta di una sola macchina virtuale per un totale di 1 *vCPU*.

Infine l'introduzione del metodo *JiTA* permette un'analisi più elastica del sistema sfruttando non solo le informazioni provenienti dal valore della *demand*, ma anche quelle dei tempi di *On* e di *Off* delle macchine virtuali. Questo modello ha evidenziato possibili miglioramenti sul carico complessivo del sistema permettendo l'eventuale allocazione di una macchina virtuale quad-core.

Capitolo 5

Conclusioni e sviluppi futuri

Il lavoro di tesi ha permesso di approfondire il funzionamento dell'algoritmo di scheduling della CPU nell'infrastruttura IaaS sviluppata da VMware. Il modello proposto, una volta generalizzato, è stato applicato per l'ideazione di due algoritmi utili alla valutazione in fase di provisioning delle macchine virtuali.

Questo capitolo riassume gli obiettivi raggiunti e i possibili sviluppi futuri che possono scaturire da questo elaborato.

5.1 Conclusioni

Il modello impostato nel Capitolo 2, grazie ai concetti di tempo di run, tempo di attesa e tempo di blocco, fornisce uno strumento utile e sufficientemente generalizzato per valutare l'impatto che ha l'allocazione di più macchine virtuali su uno stesso host. Il modello, infatti, è applicabile ad un qualsiasi sistema di virtualizzazione che prevede un algoritmo di scheduling in grado di gestire in maniera indipendente le *vCPU* della macchina virtuale e che dia la stessa priorità a tutte le *vCPU*.

Seppur non supportato da verifiche sperimentali, l'approccio generalizzato con cui è stata impostata la formulazione della teoria pone delle basi sufficientemente solide per la stima dei tempi misurati dalle CPU virtuali in altri sistemi.

I due algoritmi ideati, il *Capacity Planning Algorithm* e il *Dynamic Decision Algorithm*, forniscono ad un potenziale provider la possibilità di sfruttare a pieno la capacità di calcolo offerta dal sistema, tenendo sempre conto delle performance offerte al cliente.

L'introduzione della tecnologia *JiTA* nelle infrastrutture cloud va proprio in questa direzione. Automatizzando il processo di configurazione (scelta

dell'hardware virtuale, installazione del sistema operativo e degli applicativi) e rimandandolo al tempo effettivo di utilizzo (e quindi al tempo di allocazione sulla macchina fisica), si riesce a ricavare un certo quantitativo di risorse che, seppur minimo, permette la sovrapposizione di più ordini. Di contro il cliente ha la possibilità di risparmiare sui costi di licenza, che in un ambiente di tipo IaaS sono conteggiati in base al tempo di utilizzo, potendo *prenotare* la macchina virtuale e configurandola in attesa di un utilizzo futuro.

Non solo, sotto le ipotesi del modello del Capitolo 2, essendo le *vCPU* indipendenti, possono essere fornite al cliente come pool di risorse, dandogli la possibilità di scegliere a posteriori il numero di core con cui creare le proprie macchine virtuali.

Da notare che il *DDA*, a differenza del *CPA*, presenta un grado di maturità maggiore. Supportato dalle simulazioni effettuate ed illustrate nel Capitolo 4, ha uno spettro di analisi sufficientemente ampio che considera:

- il loro stato di *On* e di *Off*;
- il tempo di permanenza nel loro stato;
- la *demand* delle macchine in stato di *On* valorizzata secondo i rispettivi log di utilizzo.

Essendo questo progetto impostato dal punto di vista dei cloud provider, è notevole l'interesse che può scaturire dall'applicazione delle metodologie proposte. Ciò è dovuto anche dal fatto che test e simulazioni sono stati realizzati su un ambiente di produzione già proposto e installato come prodotto finale.

Scenari che prevedono l'aggiunta di più macchine virtuali sui singoli host possono essere oggetto di ulteriori sviluppi. Affinando il modello dei tempi di *Run*, *Block* e *Wait* e introducendo la possibilità di modificare la priorità per le singole macchine virtuali, si aprono scenari con una maggiore flessibilità e adattabilità alle esigenze del cliente.

5.2 Sviluppi futuri

L'elaborato ha mostrato delle soluzioni complete entro certi limiti di applicazione, primo fra tutti l'uguale priorità data alle CPU virtuali e quindi proporzionale, per ogni macchina virtuale, al numero di *vCPU*. Se alle varie macchine virtuali allocate in un host vengono impostate priorità differenti si possono evidenziare casi di studio che meritano un'analisi sicuramente più approfondita.

Il modello illustrato nel Capitolo 2 è da mettere in discussione e va riadatto secondo la nuova configurazione. Infatti il tempo di blocco di una macchina virtuale con priorità più bassa seguirà un andamento diverso da quelle con priorità maggiore. In quest'ottica è sicuramente interessante valutare la possibilità di un adattamento dinamico della priorità per far fronte a picchi improvvisi.

Un altro studio che può offrire spunti di miglioramento è il raffinamento del modello di calcolo del tempo di blocco. È interessante, infatti, analizzare i casi in cui le macchine virtuali fanno una richiesta di *demand* differente l'una dall'altra e capire se è possibile prevedere il tempo di blocco per ognuna. Combinando questo ad un adattamento dinamico della priorità si potrebbe offrire all'utilizzatore un ambiente virtuale capace di rispondere on-demand alle sue richieste.

Per quanto riguarda gli algoritmi proposti, *CPA* e *DDA*, entrambi possono essere sfruttati come base di partenza per un ampliamento delle loro funzionalità. Il *CPA*, che dà al provider la possibilità di capire quanta capacità di calcolo è in grado ancora di fornire il proprio datacenter, non considera, in questa prima versione, la possibilità di migrare le macchine virtuali tra vari host. Togliere questa limitazione può portare ad un utilizzo delle risorse più efficiente.

Per quanto riguarda il *DDA* potrebbe essere esteso attraverso una modifica dello schema a stati mostrato nella Sezione 4.3. Si può, infatti, prevedere la possibilità che una macchina virtuale, dopo un periodo di *On*, possa tornare in stato di *Off*. Questa modifica è stata parzialmente presa in considerazione per questo elaborato, ma lasciata in sospeso in quanto avrebbe complicato ulteriormente il modello presentato per l'analisi dell'impatto della tecnologia *JiTA*.

Nel complesso il lavoro ha permesso di mettere in luce diversi spunti che possono, se approfonditi, dare un disegno più completo del funzionamento e delle possibili evoluzioni delle infrastrutture cloud del prossimo futuro.

Elenco delle figure

1.1 Tipologie di hypervisor 16
1.2 Architettura VMWare vSphere 5 18
1.3 Schema d'esecuzione di 4 world disallineati. 20
1.4 Modello a stati dei world in VMWare ESXi 21
1.5 Esempio di suddivisione della capacità di calcolo con macchine virtuali di uguale *Shares* 23
1.6 Esempio di suddivisione della capacità di calcolo con macchine virtuali di *Shares* diversi 25
1.7 Modello a stati generalizzato 27

2.1 Variazione del tempo di blocco al variare del numero di macchine virtuali con 1 *vCPU* 33
2.2 Variazione del tempo di blocco al variare del numero di macchine virtuali con 2 *vCPU* 34
2.3 Variazione del tempo di blocco al variare del numero di macchine virtuali con 3 *vCPU* 36
2.4 Variazione del tempo di blocco al variare del numero di macchine virtuali con 4 *vCPU* 37
2.5 Tempo di *Run* durante il test di 4 macchine virtuali con *demand* pari al 100% 38
2.6 Tempo di blocco durante il test di 4 macchine virtuali con *demand* pari al 100% 38
2.7 Confronto tempo di blocco misurato e stima 1/2 41
2.8 Confronto tempo di blocco misurato e stima 2/4 42
2.9 Confronto tempo di blocco misurato e stima 8/16 44
2.10 Validazione modello stazione-cliente con 3 maccine virtuali . 44
2.11 Validazione modello stazione cliente con 5 e 6 macchine virtuali 46

3.1 Panoramica del *Capacity Planning Algorithm* 52
3.2 Esempio di due macchine virtuali con stessa *demand* con valori di *Run*, *Block* e *Wait* diversi 57

3.3 Esempio con *demand* diversa da quella in Figura 3.2 ma con stessi valori misurati 58
3.4 Esempi di incremento dell'istogramma con K=2, K=4, K=10 con $r^{\%} = 65\%$ e $w^{\%} \approx 0\%$ 59
3.5 Confronto del livello di servizio percepito con penalty lineare (linea rossa) e con penalty asintotica (linea verde) 67
3.6 Livello di servizio percepito (psl) ottenuto applicando diverse funzioni di penalità 70

4.1 Modello a stati dell'utilizzo di una macchina virtuale 77
4.2 Confronto del livello di servizio di *Response Time* minimo stimato e misurato con l'aggiunta di una macchina quad-core ($newVm1$) 83
4.3 Confronto del livello di servizio di *Response Time* aggregato, asl^{RT}, con l'aggiunta di una macchina dual-core 84
4.4 Confronto del livello di servizio di *Throughput* stimato e misurato per $vm1$ e $vm2$ con l'aggiunta di una macchina quad-core ($newVm1$) 85
4.5 Confronto del livello di servizio di *Throughput* stimato e misurato per $vm3$ e $newVm1$ con l'aggiunta di una macchina quad-core ($newVm1$) 86
4.6 Confronto del livello di servizio di *Throughput* aggregato, asl^{T}, con l'aggiunta di una macchina quad-core ($newVm1$) . 87
4.7 Confronto del livello di servizio di *Response Time* minimo stimato e misurato con l'aggiunta di una macchina dual-core ($newVm2$) 87
4.8 Confronto del livello di servizio di *Response Time* aggregato, asl^{RT}, con l'aggiunta di una macchina dual-core ($newVm2$) . 88
4.9 Confronto del livello di servizio di *Throughput* aggregato, asl^{T}, con l'aggiunta di una macchina dual-core ($newVm2$) . . 89
4.10 Confronto del livello di servizio di *Response Time* minimo stimato e misurato con l'aggiunta di una macchina single-core ($newVm3$) 90
4.11 Confronto del livello di servizio di *Response Time* aggregato, asl^{RT}, con l'aggiunta di una macchina single-core 90
4.12 Confronto del livello di servizio di *Throughput* aggregato, asl^{T}, con l'aggiunta di una macchina single-core ($newVm3$) . 91
4.13 Ripartizione della capacità fisica della CPU in caso di picco simultaneo 92

4.14 Esempi di funzioni di ripartizioni Gamma con media 10 e parametro k variabile 94
4.15 Variazione degli indici aggregati di *newVm1* 98
4.16 Variazione degli indici aggregati di *vm1* al variare del tempo di *On* di *vm3* 98
4.17 Variazione degli indici aggregati di *vm3* al variare del tempo di *On* di *vm1* 99

Elenco delle tabelle

2.1 Configurazione hardware dell'host 31
2.2 Confronto stima e misura del tempo di blocco medio 36
2.3 Crescita tempo di blocco con 4 macchine virtuali 40
2.4 Confronto modelli stazione-cliente 1/2, 2/4 e 8/16 45
2.5 Confronto stime e misure del tempo di blocco con 3 e 4 macchine virtuali 47
2.6 Confronto stime e misure del tempo di blocco con 5 e 6 macchine virtuali 48

3.1 Confronto del livello del servizio aggregato con $sl(1) = 0.4$ e $sl(2) = 1$ 67
3.2 Confronto del livello del servizio aggregato con $sl(1) = 0.6$ e $sl(2) = 0.8$ 68

4.1 Ripartizione della capacità fisica (% Host CPU) in caso di picco simultaneo 93
4.2 Valore dell'indice aggregato di *Response Time* di *newVm1* (4 *vCPU*) al variare del tempo di *On* di *vm1* e *vm3* 96
4.3 Valore dell'indice aggregato di *Throughput* di *newVm1* (4 *vCPU*) al variare del tempo di *On* di *vm1* e *vm3* 96

Bibliografia

[1] Cpu load script. https://github.com/ajurge/CPU_load.

[2] Jon C.R. Bennett and Hui Zhang. Wf2q: Worst-case fair weighted fair queueing. 1996.

[3] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.

[4] Angelo Marletta. Cpu limit. https://github.com/opsengine/cpulimit.

[5] Jason Nieh, Chris Vaill, and Hua Zhong. Virtual-time round-robin: An o(1) proportional share scheduler jason nieh chris vaill hua zhong. In *Appears in Proceedings of the 2001 USENIX Annual Technical Conference, June 2001.*, 2001.

[6] Confio Software. A comparison of oracle performance on physical and vmware and servers. 2011.

[7] Antonio Valles. Performance insights to intel® hyper-threading technology. https://software.intel.com/en-us/articles/performance-insights-to-intel-hyper-threading-technology, 10 2009.

[8] Björn Victor. Processes - abstracting the cpu. http://www.it.uu.se/edu/course/homepage/os/vt06/overview/processes, Uppsala University, Department of Information Technology.

[9] VMWare. Vmware® vsphere™: The cpu scheduler in vmware esx® 4.1. 2010.

[10] VMWare. The cpu scheduler in vmware vsphere® 5.1. 2013.

[11] Carl A. Waldspurger and E. William Weihl. Lottery scheduling: Flexible proportional-share resource management. 1994.

[12] Amos Waterland. stress project page. `http://people.seas.harvard.edu/~apw/stress/`.

[13] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-box and gray-box strategies for virtual machine migration. 2007.