Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica Magistrale

SIMULAZIONE DI DATACENTER E SUA VALIDAZIONE UTILIZZANDO TRACCE DI GOOGLE

Tesi di Laurea in Sistemi Complessi

Relatore:
Chiar.mo Prof.
Ozalp Babaoglu

Correlatore:
Chiar.mo Prof.
Moreno Marzolla

Presentata da:
Dennis Olivetti

Sessione 3
2012/2013

Alla mia famiglia, amici e colleghi...

Introduzione

In questo lavoro di tesi è stato svolto uno studio di simulazione volto a riprodurre il comportamento di un datacenter. Tale studio si inserisce in un progetto più ampio, avente l'obiettivo di prevedere stati critici all'interno di un dacetenter. Per riuscire in questo obiettivo è necessario avere a disposizione tracce riguardanti il funzionamento di datacenter reali, comprendenti anche informazioni sui guasti. Tracce di questo tipo al momento non sono disponibili, è quindi necessario ampliare ciò che è presente con le informazioni mancanti.

Per questo motivo è stato svolto il lavoro di tesi, che ha il compito di simulare il corretto funzionamento di un datacenter (validato mediante tracce reali) ed essere estendibile per comprendere dati riguardanti i guasti (non presenti sulle tracce reali).

Inizialmente verrà introdotto il funzionamento di alcuni datacenter esistenti, oltre che vari esempi di guasti realmente accaduti. Verrà successivamente mostrato uno studio effettuato su tracce disponibili, esponendo i motivi che hanno portato alla scelta di utilizzare, per questo lavoro, quelle rilasciate da Google. Sarà poi mostrato il modello di simulazione proposto, unito alle scelte effettuate ed alle assunzioni svolte per ovviare ad informazioni necessarie ma non presenti sulle tracce. Verranno in seguito presentati i risultati dello studio di simulazione, confrontati coi risultati ottenibili dalle tracce reali. In conclusione si mostreranno i risultati del simulatore ottenuti tentando di effettuare una sintesi delle tracce originali.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Stato dell'arte

1.1 Introduzione

1.1.1 Cos'è un datacenter

Per *datacenter* si intende un luogo in cui vengono ospitate un numero elevato di apparecchiature informatiche ed altri dispositivi ad esse collegati, tipicamente computer, infrastruttura di rete, infrastruttura di alimentazione e impianto di raffreddamento.
I datacenter sono caratterizzati da una elevata potenza di calcolo, memoria e capacità di memorizzazione dei dati. A causa dell'elevato numero di componenti presenti è frequente che si verifichino dei guasti, motivo per cui si utilizzano varie forme di ridondanza. A livello energetico si usano sorgenti multiple di alimentazione, a livello di rete collegamenti multipli di cavi e router, a livello di *storage* vengono mantenute delle repliche; i programmi in esecuzione sui datacenter sono tipicamente sistemi distribuiti e devono essere quindi progettati per resistere ai guasti, continuando a funzionare ad esempio nel caso di *crash* di alcuni processi [1]. Un esempio di datacenter è mostrato in Figura 1.1.

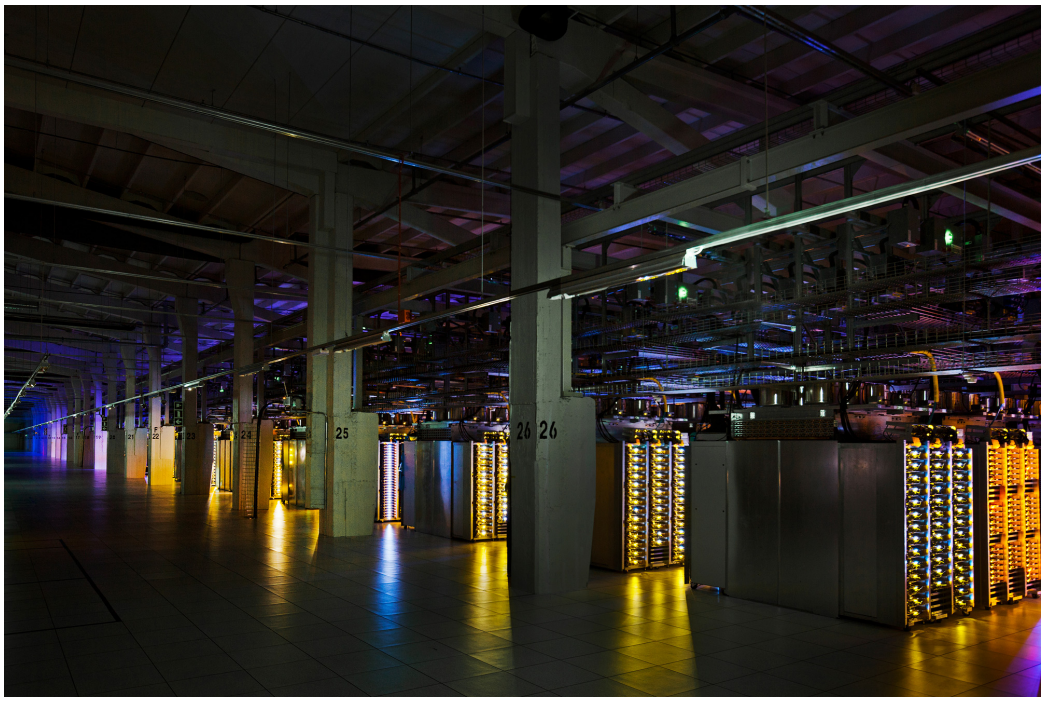

Figura 1.1: Datacenter di Google

1.1.2 I datacenter come sistemi complessi

Un sistema complesso è un sistema in cui vi sono molte componenti che interagiscono tra loro localmente, producendo un comportamento globale inatteso. Il risultato, chiamato *comportamento emergente*, non può essere individuato a priori analizzando il funzionamento delle singole componenti.

Un esempio di sistema complesso è il cervello umano, il quale è composto da molti neuroni collegati tra loro da più sinapsi. Ogni neurone interagisce in modo relativamente semplice con gli altri a cui è collegato: può essere modellato come un oggetto che riceve vari *input*, ne fa una sommatoria pesata e invia l'*output* ad altri neuroni. Si conosce quindi abbastanza bene il funzionamento dei singoli neuroni, ma il comportamento emergente di questo sistema, il cervello, è tutt'ora in fase di studio.

Molti sistemi complessi esibiscono un comportamento caotico, ovvero a piccole variazioni dei dati in input fanno corrispondere un grande cambiamen-

to nel risultato prodotto. Un esempio di sistema caotico può essere trovato nelle previsioni meteorologiche, in cui si modella la dinamica dell'atmosfera con equazioni. Queste equazioni sono esponenzialmente sensibili alla precisione dei valori dati loro in input per descrivere lo stato corrente dell'atmosfera; ciò spiega il motivo per cui non è possibile prevedere accuratamente lo stato dell'atmosfera dopo pochi giorni.

Un datacenter, per la sua dimensione, è di difficile gestione: vi sono molte macchine il cui sistema operativo e relative applicazioni vanno tenuti in costante aggiornamento, e molti servizi in esecuzione esigono un continuo controllo dei loro file di configurazione.
Aggiornare manualmente tutti i sistemi o modificare tutti i file di configurazione potrebbe quindi richiedere troppo tempo ed essere causa di errori. Oltre a questo, una modifica eseguita su un singolo componente potrebbe causare conseguenze inattese, come ad esempio un malfunzionamento globale. Questi effetti non sono prevedibili a priori, nemmeno analizzando il comportamento (più semplice) delle singole macchine facenti parte del datacenter.

Questi sono comportamenti tipici esibiti dai sistemi complessi e si vuole quindi riuscire ad analizzare un datacenter utilizzando tecniche provenienti da quest'area di studi.

1.2 Attuale gestione dei datacenter

Allo stato attuale, ogni grossa azienda gestisce il proprio datacenter con proprie metodologie. Esistono soluzioni non divulgate, come ad esempio quelle utilizzate da Google, e soluzioni disponibili al pubblico, come quelle adottate da Twitter e Facebook.
Esistono poi soluzioni utilizzate universalmente e considerate uno standard, come ad esempio Puppet, adottato da colossi come Google, Twitter, Oracle, Paypal ed altre grosse aziende.

1.2.1 Google

Il nome del sistema utilizzato da Google per gestire i loro datacenter è *Borg*. Creato nel 2003, questo sistema si occupa di scegliere su quali macchine eseguire i vari processi e di monitorare le risorse utilizzate. Nonostante funzioni adeguatamente, Google lamenta il fatto che Borg non sia in grado di gestire cluster più grandi di quelli attuali, che sia difficile aggiungere ulteriori funzionalità e che sia diventato troppo complicato [2]. Per questi motivi Google sta attualmente lavorando ad una sua evoluzione, *Omega*, con cui cerca di raggiungere i seguenti obiettivi:

- *Predicibilità*: si vuole avere un comportamento prevedibile (ad esempio nell'utilizzo delle risorse);
- *Facilità d'uso*: si vuole poter avviare processi come se si dovesse avviarli su una normale macchina;
- *Flessibilità*: deve saper resistere ai guasti (utilizzando ad esempio uno scheduling *topology-aware*[1]).

Questi obiettivi devono essere soddisfatti su larga scala e devono richiedere la minor quantità di intervento umano possibile. Più in dettaglio, per quanto riguarda il funzionamento normale, Google vorrebbe non specificare i meccanismi con cui dover soddisfare alcuni requisiti, ma solo i requisiti stessi; per quanto riguarda la gestione dei guasti l'obiettivo è quello di avere uno strumento che automaticamente capisca se si sta verificando un problema, se vi è un problema nel futuro prossimo, ed il modo giusto di reagire in caso di problemi.

Si noti che Google cerca di risparmiare più risorse possibili: non utilizza macchine virtuali ma esegue i vari processi direttamente sulle macchine fisiche. Questo viene fatto per evitare sprechi di risorse causati da un eventuale livello software aggiuntivo e per evitare di avere ulteriori sistemi operativi e applicazioni da dover amministrare ed aggiornare costantemente.

[1] Lo scheduler è a conoscenza della struttura fisica del datacenter ed esegue diverse repliche di un servizio in modo da evitare un disservizio totale in caso di guasti hardware.

1.2.2 Twitter

Twitter utilizza *Mesos* [3], un sistema sviluppato dall'università di Berkeley per avere un'alternativa open source a Borg. Questo sistema è più avanzato di Borg stesso, e si avvicina per molti versi ad Omega: cerca ad esempio di automatizzare l'utilizzo di risorse e di massimizzare la facilità d'uso.

Mesos può essere considerato il *kernel* di un datacenter: alloca risorse per altre componenti software poste sopra di esso. Alcune delle possibili componenti, ovvero quelle utilizzate anche da Twitter [4], sono:

- *Aurora*: software in grado di gestire *Long Running Service* scalabili e resistenti ai guasti;
- *Hadoop*: versione open source di *MapReduce*, sistema sviluppato da Google per fare operazioni *batch* su grosse quantità di dati presenti su un filesystem distribuito;
- *Cassandra*: DBMS distribuito, sviluppato da *Facebook* e reso successivamente open source, noto per l'assenza di *single point of failure*.

1.2.3 Facebook

Facebook ha dovuto affrontare un problema nuovo: fornire ad un miliardo di utenti una pagina web personalizzata, contenente diverse foto, messaggi e video per ognuna di esse. Nel suo sviluppo sono state affrontate difficoltà mai riscontrate prima in applicazioni del suo genere [5]. Per questo motivo sono state sviluppate nuove tecnologie, spesso adattando programmi open source esistenti. Un esempio di applicazioni utilizzate (o create appositamente) da Facebook sono:

- *Scuba*: sistema utilizzato per analizzare dati presenti in RAM su una grossa quantità di macchine;

- *Hadoop*: versione open source di *MapReduce*, sistema sviluppato da Google per fare operazioni *batch* su grosse quantità di dati presenti su un filesystem distribuito;
- *Hive*: sistema che permette di eseguire query SQL su dati Hadoop;
- *Prism*: sistema che permette di utilizzare Hadoop su diversi datacenter contemporaneamente;
- *Corona*: uno scheduler per Hadoop;
- *Peregrine*: rende Hadoop interattivo.
- *Cassandra*: DBMS distribuito noto per l'assenza di *single point of failure*.

1.2.4 Puppet

Puppet permette di gestire più sistemi contemporaneamente mediante un linguaggio descrittivo. In questo linguaggio si può ad esempio specificare che un programma debba essere installato o che un servizio debba essere sempre in esecuzione. È possibile assicurarsi anche dell'esistenza di alcuni file e del loro contenuto. Esistono configurazioni già pronte per gestire i più disparati servizi.
Oltre a questo, è possibile porre un insieme di configurazioni di vari servizi su una singola macchina, e lasciare a Puppet il compito di applicare le varie configurazioni ad ogni macchina facente parte del datacenter.
Osserviamo pertanto alcuni esempi di utilizzo di Puppet:

- *Creare un nuovo utente*: questo è possibile ottenerlo facendo diffondere a Puppet un file di configurazione in cui è presente un vincolo che richieda l'esistenza dell'utente stesso;
- *Cambiare il server DNS di ogni macchina*: in questo caso basta indicare a Puppet un vincolo sul contenuto del file adeguato.

Questo sistema, per la sua semplicità e potenza, è stato scelto da una moltitudine di aziende per gestire i propri datacenter [6].

1.3 Esempi di guasti

In questa sezione verranno mostrati alcuni casi di guasti realmente accaduti, descrivendo sia quelli gestiti correttamente, come ad esempio nel caso di Google, sia casi in cui un'errata gestione ha portato disservizi di notevole durata, come nel caso di Amazon e del Super Bowl 2013.

1.3.1 Amazon

Il caso di Amazon è un esempio in cui una piccola modifica svolta da un operatore ha causato un guasto ad un intero datacenter [7].

Un operatore cancella erroneamente una parte dei dati riguardanti l'*Amazon Elastic Load Balancing Service* (ELB), sistema utilizzato per gestire automaticamente, in base al carico, la quantità di istanze di un certo servizio da eseguire contemporaneamente. Nonostante questa modifica, tutte le applicazioni in esecuzione continuano a funzionare normalmente.

Successivamente gli utenti notano che, modificando la configurazione del proprio ELB, ottengono un degrado delle prestazioni; osservano inoltre che, creando una nuova configurazione, il tutto torna a funzionare correttamente.

Solo cinque ore dopo, Amazon riesce a capire il problema; come soluzione temporanea toglie la possibilità agli utenti di modificare il proprio ELB. Per risolvere il degrado delle prestazioni notato dagli utenti che ormai avevano modificato la configurazione, tentano di ripristinare lo stato del loro ELB a quello di poco prima la modifica eseguita dall'operatore. Questa operazione viene svolta in modo erroneo, che non solo non risolve il problema, ma causa anche una grossa perdita di tempo.

Solo dieci ore dopo Amazon riesce a ripristinare lo stato corretto per gli utenti aventi effettuato la modifica, e solo dopo altre dieci ore l'intero sistema viene sistemato completamente.

Com'è possibile che un operatore sia riuscito a mettere in ginocchio un intero datacenter? Amazon spiega che gli operatori generalmente non hanno i permessi per eseguire questo tipo di operazioni e che solitamente tali modifiche richiedono un'approvazione. In questo caso però, l'approvazione era già stata data per una modifica precedente, e la politica di Amazon era quella di salvare un'approvazione per tutte le modifiche future. Da questo guasto in poi questa politica viene cambiata, e l'approvazione viene chiesta ogni volta per ogni modifica.

Amazon è sicura di non imbattersi nuovamente in questo problema, ma di certo non può sapere quali altre modifiche possono causare problemi altrettanto disastrosi. Questo *crash*, oltre a far notare un comportamento tipico di un sistema complesso (in cui una piccola modifica locale causa un grande cambiamento globale) è un esempio di conseguenze inattese che si vorrebbe riuscire a rilevare. Un sistema in grado di capire se ci si trova in stati pericolosi, in questo caso, avrebbe potuto evitare 24 ore di downtime.

1.3.2 Google

Il caso di Google è un esempio in cui un guasto è stato rilevato e corretto automaticamente, mostrando una notevole capacità di Google nel gestire alcuni tipi di problemi [8].

Il 24 gennaio molti servizi di Google, tra cui *Gmail* e *Google+*, non sono risultati accessibili ad alcuni utenti per circa mezz'ora. La causa di questo è da attribuire a del software che, incorrendo in un bug, ha generato un'errata configurazione per alcuni servizi in esecuzione. Questi servizi hanno iniziato a comportarsi in modo erroneo, non permettendo agli utenti di accedere alla propria posta. Questo problema è stato rilevato automaticamente da servizi di controllo interni, andando ad avvisare un'apposita squadra di addetti.

Solo 12 minuti dopo, quando questa squadra stava ancora cercando di trovare la causa del problema, il sistema stesso ha capito che il guasto era stato causato dalla configurazione precedentemente generata, risolvendo automaticamente la situazione creando una nuova configurazione.

La corretta gestione di questo guasto ha portato il disservizio a durare solo 25 minuti, un notevole risultato se confrontato col tempo impiegato da Amazon per correggere il proprio guasto. Oltre a ciò questo esempio mostra come sia possibile, in alcuni casi, gestire i guasti senza l'intervento umano.

1.3.3 Superdome

Il *Super Bowl* del 2013 è un esempio di come una pessima gestione dei guasti ha portato a conseguenze non volute [9]. Si vedrà come, i principi alla base della gestione dei guasti in questo campo, siano direttamente applicabili nel caso di un datacenter.

Durante la partita viene a mancare la corrente elettrica in metà stadio per 35 minuti. La partita viene interrotta, e alla ripresa il punteggio cambia drasticamente. Questo, oltre a causare ritardi durante la diretta televisiva e una conseguente perdita di denaro, rischia di far decidere le sorti della partita ad un guasto elettrico.

Lo stadio in cui si è svolto il Super Bowl utilizzava due impianti elettrici separati per le due metà dello stadio, ognuna con un proprio salvavita. Questi salvavita non erano stati testati con un carico tipico di uno stadio, ma erano stati utilizzati con la configurazione di fabbrica.

Vediamo ora quale sarebbe stata la pratica corretta da seguire per una configurazione adeguata dell'impianto elettrico.

Evitare i guasti

I prodotti utilizzati, oltre a dover essere di buona qualità, devono essere testati in più casi:

- *Carico normale*: i salvavita non devono scattare;
- *Leggero sovraccarico*: i salvavita non devono scattare nemmeno in questo caso;

- *Sovraccarico*: i salvavita devono scattare.

Qui già si nota un primo errore svolto dai manutentori dello stadio: solo testando l'apparecchiatura avrebbero potuto evitare il problema.

Ridondanza

È importante ricorrere alla ridondanza per poter gestire il caso in cui il fornitore stesso di energia incorra in problemi e non fornisca più corrente elettrica. Nel caso dello stadio sarebbero stati necessari due generatori per coprire l'intero fabbisogno energetico.
I generatori stessi potrebbero guastarsi e sarebbe quindi prudente utilizzare un generatore aggiuntivo. Oltre a ciò, un *gruppo di continuità* è necessario per fornire corrente elettrica nel caso in cui un generatore sia guasto e quello di riserva si stia avviando.

Arginare i guasti

Se si fossero utilizzati più salvavita, ad esempio 8, sarebbe mancata la corrente elettrica per solo un ottavo dello stadio, causando sicuramente meno disguidi. Se oltre a fare ciò si fosse fatto in modo che ogni salvavita fosse responsabile di un ottavo di lampade per ogni blocco, la luce sarebbe mancata per un ottavo distribuito equamente nell'intero stadio, avendo un impatto sicuramente molto minore.

Recovery Veloce

Una volta che il salvavita è entrato in funzione, i tecnici dello stadio hanno impiegato 15 minuti per decidere se fosse sicuro fornire nuovamente l'energia elettrica. Oltre a ciò, l'illuminazione utilizzata impiegava 20 minuti a tornare alla massima luminosità. Se vi fosse stato un sistema in grado di evitare l'intervento umano e se si fossero utilizzate lampade più veloci ad accendersi, si sarebbero persi molto meno di 35 minuti.

Questi quattro metodi possono essere utilizzati all'interno di un datacenter per evitare il più possibile guasti all'impianto elettrico.

1.4 Obiettivo e motivo del lavoro

Questo lavoro di tesi si inserisce in un progetto più ampio, che ha l'obiettivo di realizzare del software che sia in grado di monitorare un datacenter e descriverne lo stato correttamente, cercando di predire se lo stato corrente è sicuro, critico o precritico. Per raggiungere questo obiettivo è importante avere a disposizione delle tracce riguardanti il funzionamento di datacenter esistenti, comprensive sia delle operazioni svolte durante il normale funzionamento, che degli eventi riguardanti i guasti. Su queste tracce si vorrebbe utilizzare strumenti riguardanti il *machine learning* per tentare di prevedere gli stati non sicuri.

Uno degli ostacoli maggiori nel raggiungere questo obiettivo è dato dal fatto che le grosse aziende tendono a non rilasciare troppe informazioni riguardanti i propri datacenter, soprattutto riguardo ai guasti. Spesso le informazioni rilasciate sono parziali, carenti delle parti più importanti o riguardanti solo l'istante prossimo al guasto. Per questi motivi, per riuscire a portare avanti comunque il lavoro, si è scelto di utilizzare tracce esistenti, aggiungendo ad esse le informazioni mancanti.
Per fare questo si è scelto di svolgere uno studio di simulazione di un datacenter, validandolo con tracce esistenti ed ampliandolo successivamente con un *modello di guasti*, in modo da poter generare nuove tracce comprendenti guasti sintetizzati.

Questo lavoro si inserisce quindi in una specifica parte del progetto più generale: è stato svolto uno studio di simulazione che modella un datacenter e ne valida il funzionamento utilizzando tracce esistenti, riguardanti il normale funzionamento del datacenter di Google.

Un lavoro simile svolto in passato è stato un progetto del 2002 chiamato Bison [10], volto ad applicare idee riguardanti i *Complex Adaptive System*

(sistemi composti da agenti che svolgono un comportamento semplice, ma che globalmente esibiscono un risultato non predicibile) ai *Network Information System* (sistemi informativi di difficile gestione, data ad esempio dall'alto numero di macchine presenti). In Bison, partendo da soluzioni presenti in natura, si è cercato di crearne di nuove non presenti a priori. Queste soluzioni sono state applicate ai *NIS* per risolvere nuovi tipi di problemi.

1.5 Tracce disponibili

Come discusso precedentemente, non sono disponibili tracce complete utilizzabili per lo studio proposto. Ne esistono però un insieme di diversa utilità.

Una raccolta di dati riguardanti i guasti è contenuta nel *Computer Failure Data Repository* [11]: un sito web che colleziona tracce riguardanti vari guasti capitati a *cluster* e *grid* di diverse aziende. Nonostante il lungo elenco di tracce presenti, non è stato trovato nulla di utile per questo lavoro, per vari motivi:

- La maggior parte dei dati riguarda guasti hardware, che non è ciò che si cerca di prevedere;
- Le tracce riguardanti i guasti software sono limitate al periodo in cui il guasto accade, non permettendo uno studio completo del normale funzionamento del sistema;
- Alcune tracce riguardano un lungo periodo di raccolta dati riguardanti i messaggi di errore del sistema, ma non comprendono informazioni riguardo ciò che viene svolto sul sistema stesso.

Per questi motivi i dati forniti dal *Computer Failure Data Repository* non sono stati utilizzati.

Recentemente Google ha rilasciato delle tracce relative ad un periodo di tempo abbastanza lungo (1 mese) [12], contenenti varie informazioni: processi in esecuzione, risorse utilizzate, vincoli di esecuzione, posizionamento dei

vari processi ed alcune informazioni molto limitate relative ai guasti. Proprio queste tracce sono state utilizzate per svolgere lo studio di simulazione. Esse sono composte da circa 200 GB di dati, molti dei quali offuscati o carenti di alcune parti, solitamente proprio le informazioni utili per capire il funzionamento del datacenter di Google. Per questo motivo è stata necessaria un'accurata analisi di queste tracce in modo da estrapolare più informazioni possibili, cercando di capire meccanismi non noti a priori riguardanti *Borg*, il gestore del datacenter di Google.

1.5.1 Tracce di Google

Le tracce di Google sono composte da vari file di tipo *csv*, contengono eventi riguardanti circa 12500 macchine e sono composte dalle seguenti tabelle:

- *Task Events*: informazioni riguardanti gli eventi dei task, comprendenti per ognuno di essi l'istante di sottomissione, l'avvio (e relativo posizionamento) e la terminazione, nonché informazioni riguardanti le risorse richieste;

- *Task Constraints*: ogni task può richiedere vincoli sulle macchine, limitando quelle su cui è possibile porlo in esecuzione. Può richiedere ad esempio che la macchina abbia almeno una certa quantità di memoria, un IP visibile dall'esterno della rete, una certa architettura software;

- *Task Usage*: per ogni task è specificata la quantità di risorse utilizzate in intervalli di 5 minuti;

- *Job Events*: i task sono raggruppati in job, questa tabella contiene informazioni già presenti in *Task Events*, più precisamente gli eventi riguardanti il primo avvio e l'ultima terminazione dei task facenti parte di un job;

- *Machine Events*: le macchine vengono aggiunte, rimosse e aggiornate, questa tabella contiene gli orari in cui questo accade, oltre agli aggiornamenti alle risorse effettuati;

- *Machine Attributes*: per ogni macchina vengono specificati gli attributi, informazioni utilizzate in *Task Constraints* per specificare i vincoli.

Molte di queste informazioni sono offuscate, ad esempio gli attributi delle macchine e i vincoli posti dai task non sono leggibili in chiaro, ma sono *hash* confrontabili fra loro. Altre informazioni sono normalizzate: tutte le risorse ad esempio sono specificate come rapporto rispetto a quelle presenti nella macchina con quantità massima. Non sono specificate informazioni riguardanti la rete in quanto questa non è considerata un collo di bottiglia, nemmeno informazioni sull'area di storage sono presenti in quanto questa è considerata essere equamente accessibile da tutte le macchine.

In letteratura sono presenti vari lavori svolti sulle tracce di Google: alcuni di essi sono stati di aiuto per ottenere informazioni utili, altri lavori sono stati utilizzati per confrontare e validare i risultati della simulazione.

Analisi statistiche

Le tracce di Google sono composte da circa 200 GB di dati, quindi non sorprende che possano esistere molti lavori che si limitano ad analizzare questi dati, estraendone alcune informazioni utili [13, 14, 15, 16, 17, 18]. Sono stati analizzati la durata media dei processi, la causa di terminazione, e l'utilizzo di risorse. Vari lavori hanno ottenuto risultati simili, concordando ad esempio sul fatto che non è possibile approssimare i dati con distribuzioni note: né ad esempio l'arrivo dei processi o l'utilizzo di risorse seguono distribuzioni classiche, come *esponenziali* o *gaussiane*. L'unico andamento simile a distribuzioni note è il tempo che intercorre tra i guasti delle macchine, che esibisce una distribuzione esponenziale inversa.

Analisi dei vincoli

Alcuni studi si sono indirizzati nell'analizzare i vincoli, posti sui processi per limitare le macchine in cui è possibile eseguirli [19]. Questi vincoli sono in parte offuscati [20], ad esempio quando viene indicato che una certa risorsa deve superare una certa quantità non è possibile sapere di quale risorsa si tratti e nemmeno conoscere la quantità in valore assoluto, ma la si conosce solo come rapporto rispetto alla massima possibile. Un altro tipo di vincolo, questa volta presente in chiaro sulle tracce, è l'*antiaffinity*: i processi facenti parte dello stesso gruppo devono essere posti in esecuzione ognuno su una macchina diversa da quella di tutti gli altri. Vi sono inoltre dei vincoli le cui informazioni sono totalmente mancanti, come ad esempio le dipendenze tra processi: questa informazione viene approssimata mostrando l'avvio del processo dipendente solo una volta che è terminato quello principale.
Questi studi hanno svolto un lavoro di *clusterizzazione*, cercando pattern che accomunino i vincoli richiesti dai vari processi. Oltre a questo sono stati confrontati i tempi di attesa medi dei processi richiedenti vincoli con quelli che non li richiedono, mostrando risultati compatibili con quelli dello studio di simulazione svolto.

Tracce come input

Diversi lavori hanno utilizzato queste tracce come dataset rappresentante il workload di un vero datacenter. In uno di essi ad esempio è stato progettato un cluster distribuito, affermando che sulle tracce di Google stesse, riesce a superarne le prestazioni [21].
In un altro lavoro è stato tentato di minimizzare le macchine inutilizzate all'interno di un *cluster*, utilizzando come input diversi esempi di workload. [22]

Previsione di workload

Alcuni studi hanno tentato di prevedere il workload di un datacenter [23]. Per fare ciò si è analizzata la fluttuazione del carico a breve e lungo termine, costruendo uno scheduler che mescolasse in modo opportuno i risultati di entrambe le previsioni. Sia in questo lavoro che in altri si è verificato che nei datacenter si ha un carico molto più variabile che nei Grid, il che rende molto più difficile prevederne il comportamento [24].

Simulazione di datacenter

Uno studio di simulazione è stato svolto da Google stesso sui dati rilasciati, più precisamente due lavori sono stati svolti parallelamente [25].

Il primo utilizza approssimazioni delle tracce reali e viene fatto uso del vero scheduler presente all'interno di *Borg*; vengono forniti i risultati di questo studio, ma non viene dato a disposizione lo scheduler stesso, non è quindi possibile utilizzare questo lavoro per altri tipi di studi.

Del secondo studio di simulazione effettuato viene rilasciato anche il codice sorgente, ma in questo lavoro sono state effettuate troppe semplificazioni, sia per abbassare i tempi di simulazione, sia per non rilasciare dettagli riguardanti le politiche interne del loro scheduler reale.
In questo secondo lavoro si suppone che tutte le macchine abbiano le stesse risorse, e che non vi siano priorità nell'eseguire i processi. Oltre a questo viene supposto che inizialmente lo scheduler conosca tutti i processi che dovranno essere eseguiti nel futuro. Per queste eccessive semplificazioni è stato scelto di non basare il lavoro sui risultati appena descritti.

Capitolo 2

Studio di Simulazione

2.1 Struttura del simulatore

2.1.1 Architettura generale

Lo studio di simulazione è stato effettuato mediante Omnet++, un simulatore *general purpose* basato su eventi. In questo simulatore è necessario implementare (in C++) le varie entità che compongono il modello e farle dialogare tra loro mediante scambio messaggi.

Sono state implementate entità attive, ovvero in grado di generare o ricevere messaggi, e entità passive, ovvero normali classi in grado di memorizzare informazioni. Fra le entità passive menzioniamo:

- *Task*: un processo pronto per essere eseguito oppure già in esecuzione, caratterizzato da un *Id* e dalle risorse che richiede;
- *Job*: un insieme di task caratterizzato da un *Id*.

Queste entità non scambiano messaggi con altre, lasciando che siano le altre entità a generare e gestire eventi legati a queste.
Le entità attive sono:

- *Macchine*: si preoccupano di gestire i task in esecuzione;
- *Arrival*: entità che si occupano di generare nuovi eventi;

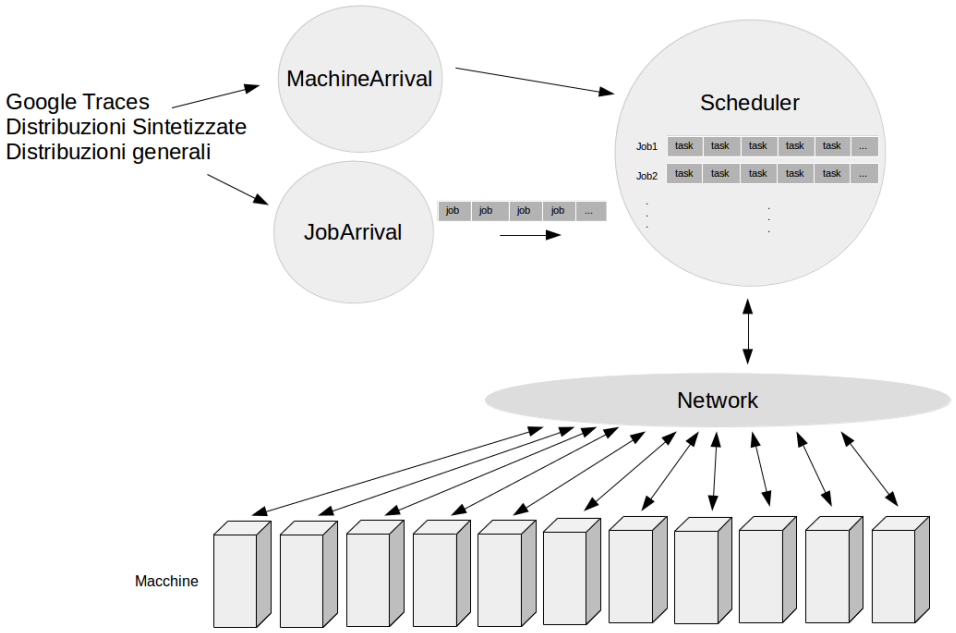

Figura 2.1: Schema di funzionamento del simulatore

- *Scheduler*: si preoccupa di gestire la coda di processi in arrivo, scegliendo di volta in volta quali Task eseguire e su quali macchine;
- *Network*: ha il compito di collegare le varie entità tra loro, permettendone la comunicazione.

In Figura 2.1 è mostrato il funzionamento generale del simulatore. Si vedrà ora una più dettagliata descrizione di ognuna di queste entità, seguita da una descrizione di come dialogano fra loro.

2.1.2 Task e Job

Un task è un'entità che non invia nè riceve messaggi. Ogni task è caratterizzato dal proprio Id, dalle informazioni riguardanti le risorse richieste e le risorse effettivamente utilizzate, da una priorità, una durata, la causa di terminazione e alcune informazioni riguardanti i vincoli di esecuzione.

Tra le risorse richieste e utilizzate vi sono la quantità di CPU e RAM: ogni task richiede allo scheduler una certa quantità (stimata dall'utente) di risorse, ma una volta che viene posto in esecuzione la quantità di risorse utilizzate è differente.

Si noti che la durata totale del task, la causa di terminazione e l'utilizzo di risorse effettivo, non sono informazioni utilizzate nella simulazione per prendere decisioni, come ad esempio la macchina su cui eseguire il processo. Queste informazioni vengono utilizzate solo nel momento in cui, in una situazione reale, risulterebbero disponibili: tale scelta implementativa è necessaria per simulare una situazione il più reale possibile.

Un job non è altro che una lista di task caratterizzata da un Id numerico. Si noti che, in linea con il funzionamento del datacenter di Goole, nel modello simulato non vi sono vincoli sul fatto che task dello stesso job debbano essere eseguiti contemporaneamente.

2.1.3 Arrival

Sono presenti due entità che si preoccupano di generare eventi e di inviarli alle altre entità della simulazione. Esse sono:

- *JobArrival*: si occupa di generare eventi riguardanti l'arrivo dei job e dei relativi task;
- *MachineArrival*: genera eventi relativi alla manutenzione delle macchine.

Più in dettaglio, *JobArrival* si preoccupa di gestire la sorgente da cui vengono ottenuti i dati riguardanti l'arrivo dei job. I dati possono essere forniti in tre diverse modalità:

- *Tracce di Google*: in questa modalità le informazioni riguardanti i job vengono caricati da un file di traccia all'avvio della simulazione;
- *Distribuzioni sintetizzate*: in questa modalità vengono utilizzate delle distribuzioni *built-in* che approssimano le tracce di Google;

- *Distribuzioni a scelta*: è possibile specificare in un file di configurazione ogni distribuzione da utilizzare per gestire, ad esempio, il numero di task per job, la quantità di risorse richieste ed utilizzate, la priorità, ed altri dati già descritti precedentemente riguardanti i task.

In ognuno di questi casi viene generato un job con le caratteristiche specificate nelle tracce o nelle distribuzioni. Successivamente questo job viene inviato allo scheduler e viene atteso un intervallo di tempo specificato nelle tracce stesse prima di ripetere il tutto. Si noti che il resto della simulazione non ha bisogno di conoscere se i dati generati da *JobArrival* provengono da tracce reali o da una distribuzione.

JobArrival si appoggia ad altre entità passive, più semplici, che si preoccupano di caricare e fornire informazioni riguardanti i vincoli di ogni task, fornibili anch'essi al simulatore mediante tracce o distribuzioni.

MachineArrival gestisce invece eventi relativi alle macchine, che possono essere forniti nello stesso modo di quelli dei job e dei task. Questi eventi riguardano la rimozione ed il ripristino delle macchine, oltre che eventi di aggiornamento, caso in cui le risorse massime di una macchina vengono alterate. In ognuna di queste circostanze la macchina viene avvisata dell'avvenuta modifica, e sarà proprio quest'ultima ad avvisare lo scheduler della propria rimozione, aggiunta o aggiornamento.

2.1.4 Macchine

Ogni macchina è incaricata di gestire i task in esecuzione, è caratterizzata da un Id e da una quantità massima di risorse allocabili su di essa.
Questa inizia la sua attività quando *MachineArrival* pone in esecuzione la macchina stessa, specificandone le risorse disponibili. Verrà notificato quindi lo Scheduler, il quale invierà alla macchina, quando necessario, i task da eseguire. Questa invierà allo scheduler ad intervalli regolari la quantità di risorse utilizzate dai task in esecuzione su di essa.

Lo scheduler non conosce in ogni istante l'esatto stato della macchina e l'esatto utilizzo di risorse di ogni task, pertanto potrebbe accadere che venga

posto in esecuzione un task che utilizza più risorse di quelle effettivamente disponibili. In questo caso la macchina deve scegliere quali task interrompere e inviare allo scheduler (*evicted* task), il quale li porrà nuovamente nella coda dei processi in attesa di esecuzione (*ready queue*).

Quando la macchina pone in esecuzione un task, genera un evento futuro: quello di terminazione del task; quando questo evento verrà gestito, lo scheduler ne sarà avvisato. Si noti che il tempo di terminazione viene utilizzato solamente per generare l'evento, non viene invece usato per prendere decisioni. Questo è necessario per simulare una situazione in cui non si conosce a priori il tempo di esecuzione di un task.

2.1.5 Scheduler

Lo scheduler si occupa di scegliere le macchine su cui porre in esecuzione i vari task. È l'entità più attiva della simulazione, riceve messaggi da varie entità e agisce di conseguenza:

- *Job in arrivo*: quando *JobArrival* invia un job, i task che lo compongono devono essere posti in esecuzione e per fare ciò lo scheduler li inserisce nella *ready queue*;

- *Stato delle macchine*: ogni macchina invia periodicamente il proprio stato (ovvero le risorse utilizzate), lo scheduler salva le informazioni ricevute, utilizzandole poi per scegliere quali task eseguire e dove;

- *Eventi delle macchine*: quando *MachineArrival* genera un evento di manutenzione, la macchina notifica lo scheduler, che salva questa informazione in modo da non inviare task a macchine al momento non disponibili;

- *Task terminati*: quando un task termina, la macchina notifica lo scheduler, il quale, se possibile, pone in esecuzione nuovi task sulla macchina;

- *Task interrotti*: quando una macchina supera le risorse disponibili, questa interrompe alcuni task e li invia allo scheduler, che li pone nella *ready queue* pronti per essere riavviati.

Lo scheduler tenta periodicamente di avviare i processi presenti nella *ready queue* e per ognuno di essi calcola quali macchine soddisfano i vincoli posti dal task. Per ogni macchina che soddisfa i requisiti controlla le risorse disponibili: se queste sono almeno quanto richiesto dal task, o se è possibile liberare risorse interrompendo processi a più bassa priorità, il task viene rimosso dalla *ready queue* e inviato alla macchina.

2.1.6 Network

Omnet permette di far dialogare entità fra loro solo se vi è un canale (*gate*) che le collega. Per permettere a tutte le entità di dialogare fra loro senza doversi preoccupare di avere un gate di comunicazione fra ogni coppia di entità, è stato posto un livello sottostante, chamato appunto *Network*, a cui è collegata ogni entità. In questo modo è possibile utilizzare un unico gate per dialogare con ognuna delle altre entità. Ogni messaggio scambiato contiene quindi due identificatori: quello del mittente e quello del destinatario. Sarà il livello Network a preoccuparsi di instradare correttamente i vari messaggi. Questa situazione rispecchia comunque la configurazione del datacenter di Google, in cui vi è una rete comune e l'area di *storage* è accessibile da ogni macchina uniformemente.

Non è stato posto un limite all'ampiezza di banda, e la latenza dei vari canali è stata posta a nulla. Questa scelta rispecchia il fatto che, almeno nel cluster di Google, non sembra essere la rete interna un collo di bottiglia. Questa entità supporta comunque una possibile futura impostazione di latenza e banda massima fra i vari canali.

Tabella	Dimensione	# Righe	# Attributi
Machine Events	3 MB	37780	6
Machine Attributes	1.1 GB	10748566	5
Job Events	315 MB	2012242	8
Task Events	15.4 GB	144648288	13
Task Constraints	2.8 GB	28485619	6
Task Usage	158 GB	1232792102	19

Tabella 2.1: Tracce di Google

2.2 Tracce di Google: struttura

2.2.1 Descrizione delle tracce

Google, nel 2011, ha rilasciato delle tracce che registrano, per un periodo di 30 giorni, il funzionamento di uno dei suoi cluster. In Tabella 2.1 sono elencate le tabelle fornite, con alcune informazioni riassuntive.

Questi dati, in generale, contengono valori offuscati o normalizzati, questo è stato fatto da Google per evitare di rilasciare troppe informazioni riguardo la loro reale potenza di calcolo, oltre che per evitare di rilasciare informazioni sensibili, come ad esempio i nomi degli utenti e il nome dei processi. Vediamo ora più in dettaglio il contenuto di ogni tabella [26].

2.2.2 Machine Events

Questa tabella contiene gli eventi di manutenzione delle macchine, comprensivi delle risorse da attribuire ad ognuna di esse. Le macchine già presenti nel cluster sono descritte come inserite al tempo zero. Gli attributi presenti sono:

- *Time*: Tempo, in microsecondi, in cui accade l'evento corrente;
- *Machine ID*: Id numerico della macchina in questione;
- *Event type*: Campo indicante se è un evento di *Up*, *Down* o *Update*;

- *Platform ID*: Stringa offuscata, confrontabile per controllare l'uguaglianza tra piattaforme;
- *CPUs*: Valore normalizzato indicante la quantità di core della CPU;
- *Memory*: Valore normalizzato indicante la quantità di memoria RAM.

I valori di CPU e memoria sono normalizzati indipendentemente da 0 a 1, dove 1 indica la capacità della macchina con la massima quantità di risorsa disponibile.

2.2.3 Machine Attributes

Questa tabella descrive come variano nel tempo gli attributi delle macchine. Questi attributi riguardano numerose caratteristiche, come ad esempio la frequenza del processore, la presenza di un collegamento con l'esterno della rete o la versione del kernel. Sia il nome dell'attributo, che il suo valore, sono però offuscati: non è quindi possibile conoscere ad esempio la frequenza dei processori, né qual è l'attributo che li descrive (non si può pertanto sapere se la frequenza del processore di una macchina è maggiore di quella di un'altra). Gli attributi numerici caratterizzati da un insieme limitato di possibili valori, vengono normalizzati ordinandone l'insieme e fornendo come nuovo valore la posizione all'interno di questo. In tal caso è possibile confrontare due valori e sapere quale è il maggiore, ma non sono presenti informazioni quantificanti la differenza stessa tra i due. I campi di questa tabella sono i seguenti:

- *Time*: Tempo, in microsecondi, in cui la macchina assume un attributo;
- *Machine ID*: Id numerico della macchina in questione;
- *Attribute name*: Stringa offuscata indicante l'attributo;
- *Attribute value*: Valore offuscato dell'attributo;
- *Attribute deleted*: Campo indicante se il valore è stato assunto o se l'attributo è stato rimosso.

2.2.4 Task Events

Questa tabella contiene ogni informazione riguardante lo scheduling dei task, dall'invio alla terminazione; per ogni task è specificata anche la quantità di risorse richieste. I campi sono i seguenti:

- *Time*: Tempo, in microsecondi, in cui accade l'evento;
- *Missing info*: flag indicante se questo evento è sintetizzato a causa di informazioni mancanti;
- *Job ID*: Id numerico del job;
- *Task Index*: Id numerico del task;
- *Machine ID*: Macchina in cui accade l'evento (ad esempio dove il task viene posto in esecuzione);
- *Event type*: Un task può essere inviato allo scheduler, avviato, terminato, ucciso, ...;
- *User*: Stringa offuscata indicante l'utente che ha avviato il task;
- *Scheduling class*: Valore indicante quanto questo task è sensibile alla latenza;
- *Priority*: Priorità del task;
- *CPU request*: Quantità di CPU richiesta;
- *Memory request*: Memoria richiesta;
- *Disk space request*: Quantità di spazio disco richiesto;
- *Different machine restriction*: Campo indicante se questo task chiede il vincolo *antiaffinity*.

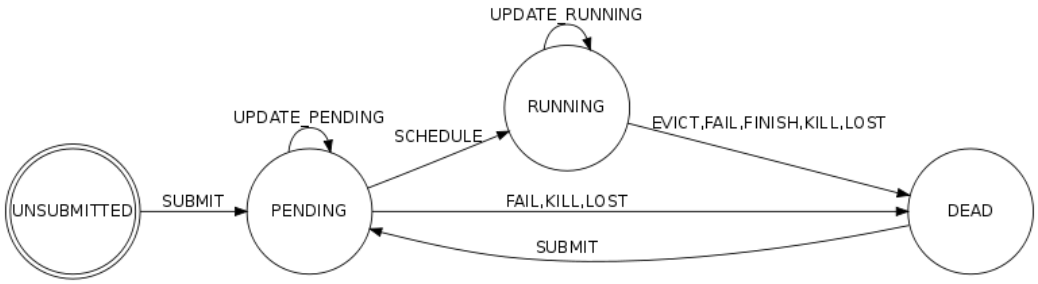

Figura 2.2: Schema di funzionamento dello scheduler

Si noti che le quantità di CPU, RAM e spazio disco richieste sono in proporzione al valore normalizzato spiegato precedentemente. Inoltre si osservi che, nonostante sia indicato il volume di spazio disco richiesto, nelle tracce non è presente la quantità disponibile in ogni macchina. In Figura 2.2 sono mostrati gli stati che un task può assumere.

2.2.5 Job Events

Questa tabella contiene un sottoinsieme dei dati presenti in *Task events*; nel caso un job venga avviato, qui è presente l'evento associato al primo task del gruppo, mentre nel caso di terminazione viene riportata solo la terminazione dell'ultimo task facente parte del job. L'unica informazione aggiuntiva disponibile in questa tabella è una stringa offuscata rappresentante il nome del job stesso.

2.2.6 Task Constraints

Questa tabella contiene i vincoli posti da ogni task: viene specificato un attributo della macchina, un valore dell'attributo e un operatore di confronto. Questi valori sono offuscati nello stesso modo in cui lo sono in *Machine attributes*, è quindi possibile confrontare il valore richiesto dal task con il valore presente in ogni macchina. Più in dettaglio, la tabella contiene i seguenti attributi:

- *Time*: Tempo, in microsecondi. Un task può comparire più volte nelle tracce, chiedendo vincoli diversi;
- *Job ID*: Id numerico del job;
- *Task index*: Id numerico del task;
- *Comparison operator*: valore indicante se la quantità richiesta deve essere maggiore, minore, uguale o diversa rispetto a quella della macchina;
- *Attribute name*: Stringa offuscata indicante l'attributo;
- *Attribute value*: Stringa offuscata o numero indicante il valore dell'attributo.

2.2.7 Task Usage

In questa tabella sono presenti informazioni indicanti l'utilizzo di risorse dei task in intervalli di 5 minuti. Queste informazioni comprendono il reale utilizzo (medio e massimo) di memoria e di CPU, oltre che dettagliate informazioni riguardanti l'accesso al disco e alla memoria.

2.3 Come sono state utilizzate le tracce di Google

Le tracce descritte precedentemente non sono state direttamente utilizzate nel simulatore, ma sono state modificate, semplificandone la struttura e tenendo solo le informazioni necessarie. Più in dettaglio:

- Sono state rimosse informazioni non utilizzate nella simulazione, come ad esempio i nomi utenti e i nomi dei job;

- I vincoli sono stati trasformati da stringhe offuscate a valori numerici (per velocizzare la simulazione in caso di confronto di vincoli, oltre che per utilizzare meno memoria);

- Le informazioni riguardanti l'utilizzo di risorse dei task sono state semplificate;

- Sono stati rimossi eventi riguardanti le scelte effettuate dallo scheduler di Google, come ad esempio l'orario di avvio dei task e le macchine su cui i task vengono posti. Il simulatore deve riuscire ad ottenere risultati simili, e quindi a comportarsi come lo scheduler di Google, senza conoscere a priori questi eventi.

A partire dalle 6 diverse tracce rilasciate da Google, sono state quindi generate 4 tracce utilizzate dal simulatore. Si vedrà ora più in dettaglio come è stata generata ognuna di esse.

Eventi delle macchine Questa tabella è una semplificazione della tabella *Machine Events* di Google. È stato rimosso il *Platform Id*; gli Id delle macchine sono stati mappati nell'intervallo compreso tra 0 e il numero di macchine totali.

Attributi delle macchine Questa tabella è una semplificazione della tabella *Machine Attributes* di Google. In essa sono state effettuate alcune semplificazioni:

- Si suppone che gli attributi delle macchine non cambino nel tempo (questo oltre a semplificare la simulazione, non incide nei risultati di essa in quanto gli eventi di questo tipo sono una minoranza);

- Gli Id delle macchine sono stati mappati nell'intervallo compreso tra 0 e il numero di macchine totali.

- Sono stati rimossi gli attributi non utilizzati nei vincoli dei task;

- Gli attributi sono stati trasformati, da stringhe, in valori numerici.

Si ottiene quindi una tabella di tre colonne: Id della macchina, Id del vincolo, valore numerico del vincolo.

Tracce Task e Job In questa tabella sono contenute tutte le informazioni riguardanti task e job, sono presenti quindi informazioni fornite da *Job Events*, *Task Events* e *Task Usage*. I campi di ogni riga sono:

- *Time*: Tempo in cui il task viene inviato allo scheduler;
- *Job Id*: Id del job;
- *Task Id*: Id del task;
- *Scheduling class*: sensibilità alla latenza del task;
- *Priority*: priorità richiesta dal task;
- *Requested CPU*: quantità normalizzata di CPU richiesta (fornita da *Task Events*);
- *Requested RAM*: quantità normalizzata di memoria richiesta (fornita da *Task Events*);
- *Requested Disk*: quantità normalizzata di spazio disco richiesto (fornita da *Task Events*);
- *Duration*: durata del task, di questo argomento si parlerà più in dettaglio successivamente;
- *Causa di terminazione*: un task può terminare normalmente o essere ucciso da un utente;
- *Antiaffinity*: campo indicante se task dello stesso job vadano eseguiti su diverse macchine;

- *CPU usage*: utilizzo reale medio normalizzato di CPU (media dei valori presenti in *Task Usage*);
- *RAM usage*: utilizzo reale medio normalizzato di memoria (media dei valori presenti in *Task Usage*);
- *Disk usage*: utilizzo reale medio normalizzato di spazio disco (media dei valori presenti in *Task Usage*);
- *Disk time usage*: frazione di tempo utilizzata nell'accesso al disco (media dei valori presenti in *Task Usage*);

La durata di ogni task è stata calcolata in seguito ad alcune osservazioni:

- Se un task viene interrotto (*evicted*) per far spazio ad un altro, il task interrotto viene fatto ripartire dal suo stato iniziale (questo comportamento si può notare durante l'esecuzione di task di tipo *MapReduce*);
- Se la causa di terminazione di un task è un *kill*, solitamente si tratta di processi con un lungo tempo di esecuzione (come ad esempio server) che non sono pensati per svolgere dei calcoli per poi terminare, ma per restare in esecuzione finché l'utente lo decide; un'altra possibile causa di kill può essere un errato avvio, solitamente seguita da un riavvio dei task in un successivo momento.

Per questi motivi si è scelto di considerare come durata, nel caso di un task *terminante*, il tempo che intercorre tra il suo ultimo riavvio e la sua terminazione naturale; si è scelto invece di considerare come durata di un task *ucciso*, il tempo che intercorre tra il suo *submit* e il suo evento di *kill*.

Si noti che, in ogni caso, durante la simulazione, si suppone di non conoscere né la causa, né la durata, per prendere decisioni. Queste informazioni sono utilizzate solo per generare i relativi eventi di terminazione, che vengono creati differentemente a seconda della causa: in caso di terminazione naturale viene generato un evento che dipende dall'orario corrente e dalla durata, invece nel caso di terminazione forzata l'evento dipende dall'orario

di submit e dalla durata. Questo implica che i task a terminazione forzata terminino indipendentemente dal loro tempo di esecuzione effettivo. Tale semplificazione è necessaria dato che non è in nessun caso possibile conoscere il motivo che ha portato l'utente a terminare forzatamente i propri task.

Constraint Questa tabella contiene i vincoli posti dai task; gli attributi e i valori richiesti sono stati modificati nello stesso modo descritto precedentemente nel caso degli attributi delle macchine. Sono presenti cinque colonne: job e task Id, operatore di confronto, valore numerico dell'attributo e del suo valore.

2.3.1 Informazioni mancanti e semplificazioni

Alcune informazioni necessarie per capire le decisioni prese dal reale scheduler di Google non sono presenti nelle tracce: ad esempio, nonostante sia noto dalla letteratura che la risorsa che costituisce collo di bottiglia sia la memoria, non è noto il motivo per cui il suo utilizzo stia mediamente (e stabilmente) a valori piuttosto bassi (circa il 50%). La spiegazione di Google (nel non fornire questa informazione) sta nel fatto che parte delle risorse sono utilizzate per il sistema stesso, o per altri processi. Per questo motivo, nel modello simulato, è stato utilizzato come valore massimo di memoria utilizzabile per ogni macchina il valore di utilizzo medio ottenibile dalle tracce.

Un'altra informazione non presente è l'utilizzo di memoria di un task nel tempo immediatamente successivo all'avvio (la granularità delle informazioni fornite da Google è nell'ordine dei 5 minuti). Oltre a ciò, si può notare dalle tracce che, una volta avviato un task, sebbene sia disponibile ulteriore memoria sulla macchina (stando all'utilizzo reale del task), Google tende ad aspettare del tempo prima di eseguire altri task sulla stessa macchina. Non è possibile conoscere la causa di questo comportamento, si può supporre però che Google attenda alcuni secondi in modo da conoscere più precisamente le reali risorse utilizzate dal processo (all'avvio conosce solo quelle richieste/stimate). Questa situazione è stata simulata facendo in modo che

l'utilizzo di un task, appena avviato, sia quello richiesto, e solo dopo un certo tempo (definito mediante un parametro) venga utilizzata l'informazione reale. Questo permette di simulare una situazione il più verosimile possibile in cui l'utilizzo reale di risorse non è immediatamente disponibile all'avvio di un processo.

Un'altra informazione non fornita da Google è la politica di scelta dei task da interrompere per far spazio a quelli di più alta priorità. Anche in questo caso si è semplificata la situazione: non si è utilizzata l'informazione di sensibilità alla latenza, e si è ridotto a 3 il numero di classi di priorità. Questo rispecchia studi di altri lavori, in cui le priorità sono suddivise in *gratis*, *other* e *production*.
La politica con cui viene scelto in quali casi interrompere i task e quali tra loro è la seguente: lo scheduler, nello scegliere dove porre in esecuzione un task, calcola la quantità di memoria libera di ogni macchina con la seguente formula:

$RAM_f = RAM_{rf} + RAM_{lu} * param$

dove RAM_f è la memoria considerata libera (*free*), RAM_{rf} è la vera quantità di memoria libera (*real free*) e RAM_{lu} è la quantità di memoria utilizzata dai task in esecuzione sulla stessa macchina ma con più bassa priorità (*lower usage*). Più il parametro è alto, più lo scheduler diventa aggressivo.
Quando una macchina riceve un task, e va in sovraccarico nell'utilizzo di risorse, sceglie di interrompere, tra i processi a più bassa priorità, quelli avviati più recentemente. Questa politica tende a non interrompere processi che potrebbero essere vicini alla terminazione.

2.4 Sintesi di tracce

2.4.1 BiDAl

BiDAl è un programma in grado di analizzare Big Data ed estrarre statistiche utili. Permette di operare su grosse quantità di dati sia mediante query SQL che mediante linguaggi più classici, quali Hadoop ed R. Forni-

sce un insieme di comandi già pronti per essere utilizzati durante l'analisi, questo facilita il lavoro perché non necessita un'attività di programmazione. È inoltre in grado di analizzare delle tracce e ricavarne delle funzioni che ne approssimino la distribuzione dei dati. Questo software è stato utilizzato sia per approfondire la comprensione delle tracce, analizzandone vari aspetti, che per effettuare la sintesi delle tracce stesse, in modo da utilizzare come fonte di eventi le funzioni sintetizzate invece che le tracce reali.

I dati delle tracce sono stati manipolati mediante un insieme di comandi SQL ed R. In questo modo sono stati estrapolati sia valori numerici (come ad esempio il rapporto tra task *terminanti* e *uccisi*) che distribuzioni. BiDAl, nel caso delle distribuzioni, è in grado di generare del codice in linguaggio C direttamente utilizzabile all'interno del simulatore. Questo codice rappresenta una distribuzione di tipo *ECDF* (Empirical Cumulative Distribution Function): una funzione che dato in input un valore restituisce la probabilità che la distribuzione abbia un valore minore o uguale a quel valore.
I parametri estratti sono:

- probabilità che un task richieda il vincolo di antiaffinity;
- probabilità che un task richieda un generico vincolo;
- probabilità che una macchina soddisfi i vincoli;
- quantità di task iniziale;
- probabilità che un task sia *long running* (ovvero che resti in esecuzione dall'inizio della simulazione alla terminazione);
- quantità di RAM effettivamente utilizzata sulle macchine
- probabilità che un task sia *terminante* o *ucciso*.

Mentre le distribuzioni estratte sono:

- quantità di CPU richiesta dai task;
- quantità di memoria richiesta dai task;

- priorità di ogni task;
- durata dei task terminanti;
- durata dei task uccisi;
- numero di task per job;
- tempo di interarrivo dei job;
- tempo di interarrivo dei guasti delle macchine;
- durata dei guasti delle macchine;
- quantità di CPU delle macchine;
- quantità di memoria delle macchine.

2.4.2 Modifica al simulatore

Il simulatore è stato modificato affinché fosse in grado di ottenere gli eventi dalle distribuzioni invece che dalle tracce. Per fare questo sono state modificate le classi di *Arrival* inserendone all'interno le distribuzioni generate e facendo in modo che il resto del codice fosse indipendente dalla fonte degli eventi.

Non potendo più utilizzare i vincoli reali, la gestione di questi è stata semplificata: sono state analizzate le tracce al fine di studiare l'influenza dei vincoli, calcolando la percentuale di task richiedenti vincoli e la soddisfacibilità media di questi da parte delle macchine.
Ad ogni macchina è poi stato associato un valore numerico compreso in un certo intervallo, ad ogni task richiedente vincoli è stato quindi associato il vincolo di chiedere un valore compreso in un certo sottointervallo. Il rapporto fra i due intervalli è dato dalla probabilità di soddiscacibilità precedentemente calcolata. In questo modo ogni vincolo sarà soddisfatto con la stessa probabilità rilevata dalle tracce.

Capitolo 3

Utilizzo del simulatore

In questa sezione verrà mostrato il funzionamento del simulatore e verranno spiegati i possibili parametri del file di configurazione.

La configurazione è composta da due file, chiamati *omnetpp.ini* e *package.ned*. Il primo è il file di configurazione vero e proprio, editabile per cambiare i parametri della simulazione; il secondo file (da non modificare) contiene i valori di default dei vari parametri, oltre che la definizione dei vari blocchi (le componenti attive) del simulatore.

Vediamo ora un esempio di configurazione:

```
[Config traces]
network = Cluster
Cluster.num_machines = 13000
Cluster.arrival_real = 1
Cluster.tracetask = "/home/me/task_trace.csv"
Cluster.tracetaskconstraints = "/home/me/TC.csv"
Cluster.tracemachineattributes = "/home/me/MA.csv"
Cluster.tracemachineevents = "/home/me/ME.csv"
Cluster.time_max = 150000
repeat=10
```

Con questi comandi si definisce il nome della configurazione ("traces"), si specifica il numero di macchine massime, si sceglie di utilizzare le tracce rea-

li come fonte di dati, si specifica la locazione delle tracce e si da un limite temporale alla simulazione, posto a circa 40 ore. Viene specificato anche che la simulazione è composta da 10 run differenti (Omnet++ sceglie automaticamente, per ogni run, differenti semi per i numeri casuali). Per avviare la simulazione si può utilizzare il comando:

```
$ for i in ‘seq 0 9‘; do
>   ./Cluster -u Cmdenv -c traces -r $i
> done
```

Per utilizzare invece le distribuzioni sintetizzate, già incluse nel simulatore, si può utilizzare una configurazione di questo tipo:

```
[Config synth]
network = Cluster
Cluster.num_machines = 13000
Cluster.arrival_real = 0
```

In questa configurazione non è stato specificato il numero di run da eseguire e di default viene effettuato un singolo run. Si può avviare la simulazione mediante il comando:

```
$ ./Cluster -u Cmdenv -c synth -r 0
```

Per utilizzare invece distribuzioni a scelta, è possibile procedere nel seguente modo:

```
[Config custom]
network = Cluster
Cluster.num_machines = 13000
Cluster.arrival_real = 0
Cluster.distr_job_arrival = exponential(3s)
Cluster.distr_machine_downtime = truncnormal(1200s,300s)
```

In questo caso è stato specificato che i job arrivano con una distribuzione esponenziale di media 3 secondi, e che il downtime delle macchine segue una

distribuzione gaussiana con media 1200 secondi e varianza 300 secondi (in cui sono eliminati i valori negativi). In questo caso la simulazione è avviabile col seguente comando:

```
$ ./Cluster -u Cmdenv -c custom -r 0
```

Una lista di possibili distribuzioni è reperibile dal manuale di Omnet++ [27]. Una lista completa di parametri configurabili è possibile ottenerla dal file *package.ned*, in cui sono presenti i valori di default impostati a quelli estratti dalle tracce reali.

Capitolo 4

Risultati

4.1 Introduzione

Il simulatore è stato validato confrontando i risultati ottenuti con quelli osservabili dalle tracce. Sono state considerate tre metriche:

- *Processi in esecuzione*: il numero di questi processi è stato utilizzato per verificare che le risorse disponibili nelle macchine e l'utilizzo di risorse reali dei processi siano stati approssimati correttamente;
- *Processi completati*: il numero di questi processi è stato utilizzato per verificare che il simulatore si comporti correttamente globalmente, ovvero che siano corrette le scelte di approssimazione sulla durata dei processi;
- *Dimensione della ready queue e processi evicted*: questi due parametri permettono di capire quanto simile sia la politica di scheduling del modello simulato con quella di Google nello scegliere quali processi eseguire, su quali macchine, e quali processi interrompere.

Sono stati eseguiti 10 run per ogni configurazione, in modo da ottenere risultati statisticamente significativi. Essendo il simulatore quasi deterministico (l'unica variazione è data durante la scelta della macchina su cui porre il processo), gli intervalli di confidenza risultano di grandezza minima.

All'avvio il simulatore pone nella ready queue tutti i processi che sul cluster reale sono già in esecuzione, impiegando vari minuti per stabilizzarsi in uno stato simile a quello delle tracce originali (il simulatore impiega del tempo per posizionare sulle macchine tutti i processi che sulle tracce reali sono già in esecuzione). Questa fase viene considerata *transiente iniziale* ed è stata eliminata dai risultati. Gli eventi dati in input ad ogni run corrispondono a quelli riguardanti le prime 40 ore delle tracce originali, dove sono poi stati eliminati i risultati delle prime 5 ore, facenti parte del transiente iniziale.

Si è poi confrontato il tempo medio di attesa nella ready queue dei processi richiedenti vincoli di posizionamento, con quello dei processi che non li richiedono, ottenendo risultati simili con quelli già presenti in letteratura.

4.2 Processi in esecuzione

Per verificare che la gestione delle risorse venga effettuata correttamente, e per verificare che le approssimazioni effettuate sulle tracce originali non siano troppo riduttive, sono stati confrontati il numero di processi in esecuzione nel tempo sulle macchine reali e sulle macchine simulate. Come precedentemente discusso, l'utilizzo reale di risorse è stato calcolato mediante una media dell'utilizzo nel tempo dei dati presenti sulla traccia *Task Usage* rilasciata da Google, e la capacità delle macchine è stata scelta essere la metà di quella indicata dalle tracce, in accordo con risultati della letteratura. Oltre a ciò sono state eliminate le informazioni riguardanti l'aggiornamento degli attributi delle macchine.

Nelle Figure 4.1 e 4.2 si può notare come varia nel tempo il numero di task in esecuzione: nella prima vengono mostrati i dati grezzi e nella seconda, per mostrarne meglio l'andamento, ne viene fatta una media esponenziale nel tempo. Viene presentata una media dei dati raccolti in 10 run confrontata coi dati raccolti direttamente dalle tracce. Si può notare una notevole similitudine tra le due curve.

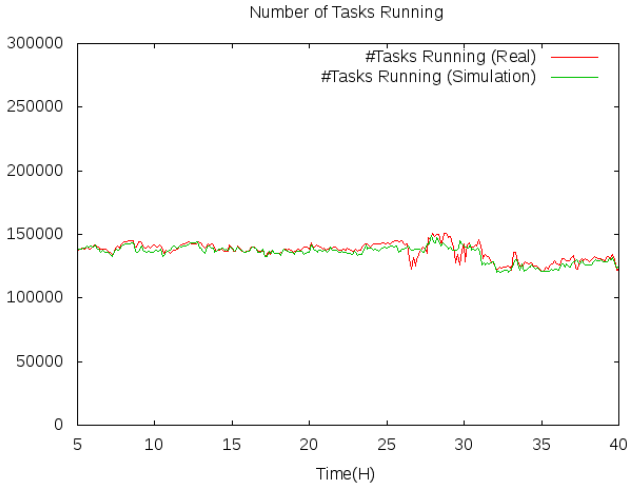

Figura 4.1: Task in esecuzione nel tempo

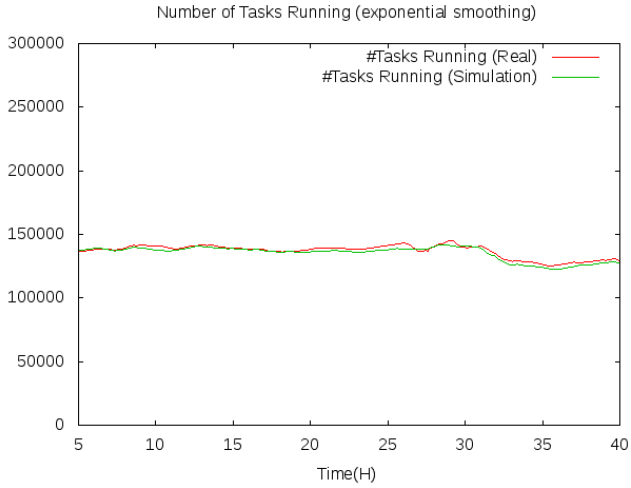

Figura 4.2: Task in esecuzione nel tempo (exponential smoothing)

Media Simulazione	134476
Media Reale	136152
Errore Massimo (valore assoluto)	4622
Errore Massimo (in percentuale rispetto alla media)	3.4%
Errore Medio (valore assoluto)	1858
Errore Medio (in percentuale rispetto alla media)	0.01%
Intervallo di Confidenza dell'errore medio (valore assoluto)	132
Intervallo di Confidenza dell'errore medio (percentuale)	7.1%

Tabella 4.1: Statistiche dei task in esecuzione in intervalli di 450 secondi.

In Tabella 4.1 si possono osservare alcune statistiche riguardanti le due curve, come gli errori massimi e medi. Per quest'ultimo è inoltre riportato l'intervallo di confidenza del 95% di tipo *T-Student*.

4.3 Processi completati

Sono state effettuate decisive semplificazioni sui dati riguardanti la durata dei processi. I processi sono stati divisi in due gruppi: quelli *terminanti* e quelli *uccisi*. Dei processi terminanti, ovvero quelli per i quali esiste sulle tracce un evento di corretta terminazione, è stata scelta come durata quella che sulle tracce originali intercorre fra l'ultimo riavvio e la corretta terminazione. Per i processi uccisi invece, ovvero quelli che terminano a causa di un *kill* effettuato da un utente, ne è stata scelta la durata come il tempo che intercorre fra il *submit* e la terminazione forzata.

Nelle Figure 4.3 e 4.4 si può notare come varia nel tempo il numero di task completati. Anche in questo caso vengono mostrati i risultati grezzi ed una loro media esponenziale nel tempo di dati raccolti in 10 run, confrontati coi dati raccolti direttamente dalle tracce. Si può notare che, eccetto lievi differenze locali, l'andamento delle due curve risulta molto simile.

Anche in questo caso vengono mostrati, in Tabella 4.2, gli errori massimi e medi (e per quest'ultimo il relativo intervallo di confidenza).

Media Simulazione	3671.3
Media Reale	3654.6
Errore Massimo (valore assoluto)	1974
Errore Massimo (in percentuale rispetto alla media)	56%
Errore Medio (valore assoluto)	246
Errore Medio (in percentuale rispetto alla media)	7%
Intervallo di Confidenza dell'errore medio (valore assoluto)	34.2
Intervallo di Confidenza dell'errore medio (percentuale)	13.9%

Tabella 4.2: Statistiche dei task completati in intervalli di 450 secondi.

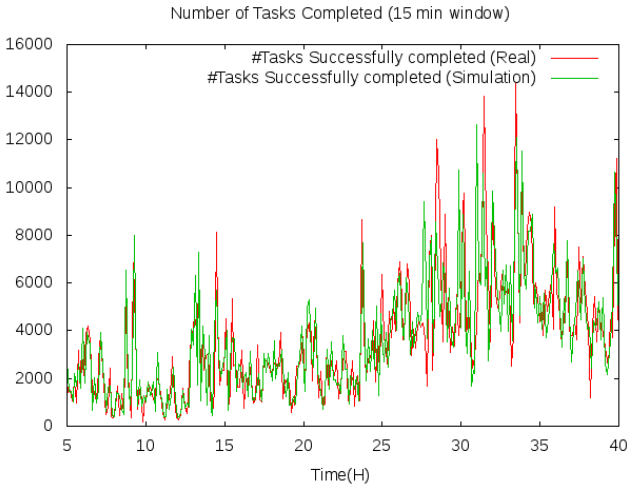

Figura 4.3: Task completati nel tempo

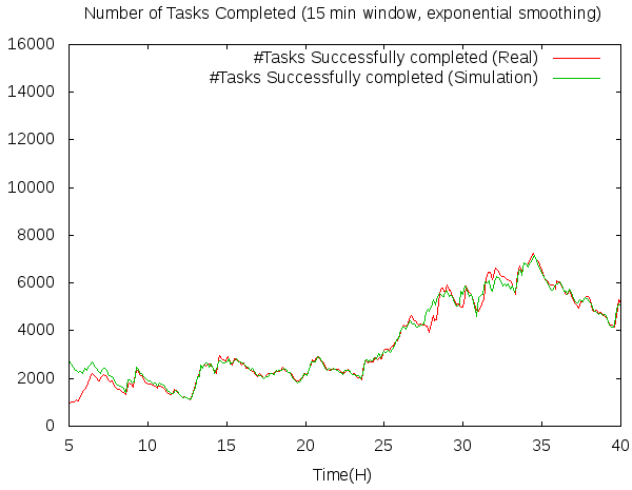

Figura 4.4: Task completati nel tempo (exponential smoothing)

4.4 Dimensione della *ready queue* e quantità di processi *evicted*

La dimensione della *ready queue* e la quantità di processi *evicted* sono state analizzate per verificare la similitudine tra la politica di scheduling del modello simulato rispetto alla politica reale (sconosciuta) dello scheduler di Google. Si noti che queste due metriche sono intrinsecamente correlate: tentando di eseguire nuovi processi interrompendone altri, da un lato vengono rimossi dalla ready queue i task posti in esecuzione, dall'altro quelli interrotti devono essere posti nuovamente in essa.

È importante ribadire la politica adottata nella simulazione: per ogni processo in attesa di essere eseguito, si calcola la lista di macchine che soddisfano i vincoli del processo e se ne cerca una con una quantità di risorse libere almeno pari a quella specificata dall'utente per il task. Le risorse a disposizione vengono calcolate considerando come libere quelle utilizzate dai task con priorità minore rispetto a quella del task di interesse. Oltre a ciò, quando un task viene posto in esecuzione, per un certo periodo di tempo il

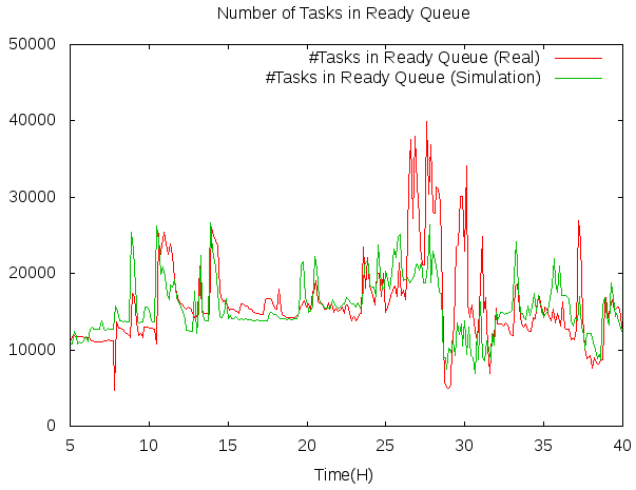

Figura 4.5: Dimensione della ready queue nel tempo

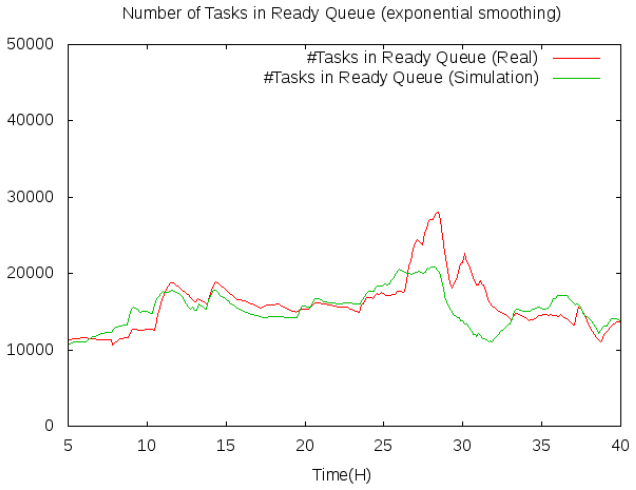

Figura 4.6: Dimensione della ready queue nel tempo (exponential smoothing)

suo utilizzo di risorse viene considerato essere quello richiesto, e non quello reale.

Nelle Figure 4.5, 4.7, 4.6 e 4.8 sono mostrati rispettivamente l'andamento della dimensione della ready queue nel tempo, l'andamento del numero di processi evicted nel tempo, e gli stessi dati mostrati mediante una media esponenziale. I dati provengono da 10 run e ne viene mostrata una media.

Si può notare che, oltre ad essere l'andamento generale delle curve molto simile, esse in molti casi presentano gli stessi picchi, indicanti una simile gestione di grosse quantità di task contemporaneamente in arrivo. La differenza più sostanziale, notabile fra le 25 e 30 ore, potrebbe essere causata dal fatto che Google considera più stringenti i vincoli di priorità e di sensibilità alla latenza, ma non è comunque possibile sapere con esattezza la causa dato che la reale politica dello scheduler di Google è sconosciuta.

Vengono quindi mostrate (nelle Tabelle 4.3 e 4.4) alcune statistiche riguardanti le due curve, come gli errori medi e massimi. Per l'errore medio è mostrato l'intervallo di confidenza del 95% di tipo *T-Student*.

Per verificare ulteriormente la bontà della simulazione, sono stati calcolati

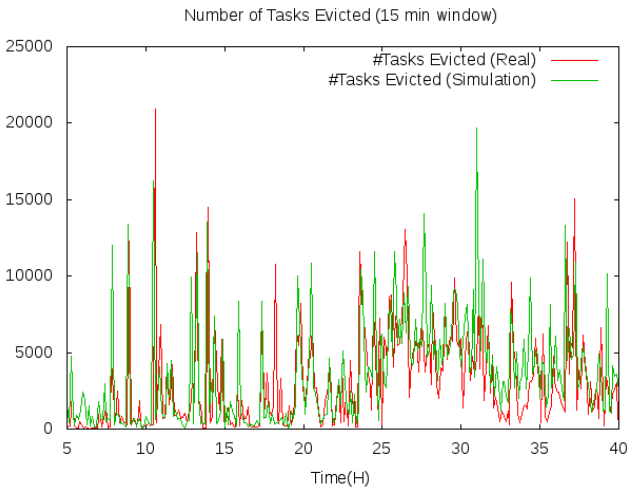

Figura 4.7: Processi evicted nel tempo

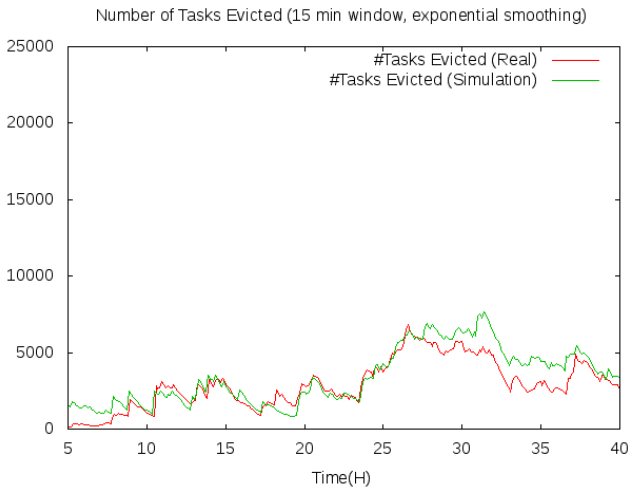

Figura 4.8: Processi evicted nel tempo (exponential smoothing)

Media Simulazione	15400.6
Media Reale	15893.9
Errore Massimo (valore assoluto)	9318
Errore Massimo (in percentuale rispetto alla media)	59%
Errore Medio (valore assoluto)	1944
Errore Medio (in percentuale rispetto alla media)	12.2%
Intervallo di Confidenza dell'errore medio (valore assoluto)	229
Intervallo di Confidenza dell'errore medio (percentuale)	11.8%

Tabella 4.3: Statistiche dei task in ready queue in intervalli di 450 secondi.

Media Simulazione	3671.32
Media Reale	2895.76
Errore Massimo (valore assoluto)	2639
Errore Massimo (in percentuale rispetto alla media)	92%
Errore Medio (valore assoluto)	755
Errore Medio (in percentuale rispetto alla media)	26%
Intervallo di Confidenza dell'errore medio (valore assoluto)	66.9
Intervallo di Confidenza dell'errore medio (percentuale)	8.9%

Tabella 4.4: Statistiche dei task evicted in intervalli di 450 secondi.

Media Simulazione	170.0
Media Reale	161.5
Dimensione intervallo di confidenza valore simulato (valore assoluto)	3.54
Dimensione intervallo di confidenza valore simulato (percentuale)	2.08 %
Differenza valore reale e simulazione	5.27%

Tabella 4.5: Tempo di attesa medio, dati reali e simulazione

i tempi di attesa medi dei task nella ready queue, rilevando una notevole similitudine tra i risultati e i dati reali, come si può notare in Tabella 4.5.

4.5 Attesa a causa di vincoli

Per validare ulteriormente lo studio di simulazione, si è cercato di confrontare i risultati con altri presenti in letteratura. Dato che la maggior parte dei lavori già presenti si limita a fare un'analisi statistica delle tracce, non è stato possibile avere molte fonti di paragone. L'unico studio presente con cui è possibile confrontare questo lavoro riguarda l'analisi dei vincoli [19]. Esso sostiene che i task richiedenti vincoli attendono circa dalle 2 alle 6 volte più tempo rispetto a quelli che non li richiedono. Questo risultato è compatibile coi risultati della simulazione: in Figura 4.9 si può notare che nella simulazione i processi richiedenti i vincoli attendono circa il triplo rispetto agli altri. Sono mostrati gli intervalli di confidenza del 95% di tipo *T-Student*, calcolati su dati raccolti da 10 run.

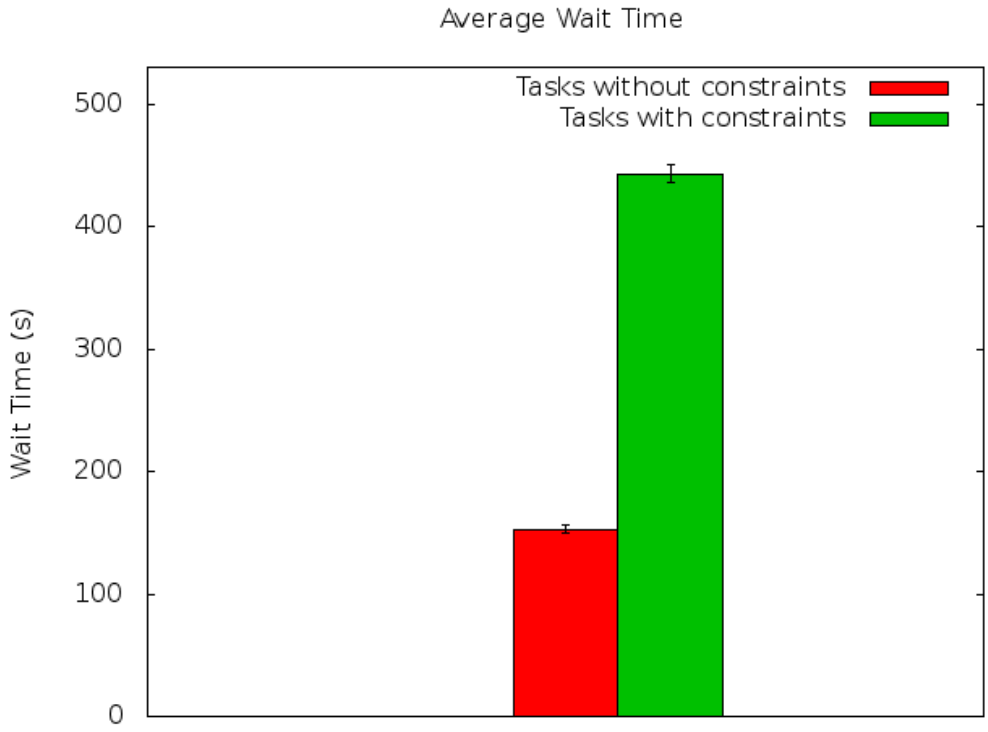

Figura 4.9: Tempo di attesa medio, task liberi e con vincoli

4.6 Risultati con dati sintetici

Una volta validato lo studio di simulazione sulle tracce reali si è cercato di capire quanto fossero corrette le distribuzioni sintetizzate. Anche in questo caso sono stati confrontati il numero di task in esecuzione, il numero di task completati, la dimensione della *ready queue* ed il numero di processi *evicted*. In questo caso però sono stati confrontati solo i valori aggregati delle curve, non potendo avere, mediante distribuzioni sintetizzate, lo stesso andamento temporale.

I risultati della simulazione sono mostrati in Tabella 4.6. La lieve differenza nel numero di task in esecuzione (e la dimensione della *ready queue*) fa notare come le distribuzioni approssimino ottimamente le risorse richieste dai task e le risorse disponibili sulle macchine, dato che il numero di task che è possibile eseguire contemporaneamente dipende strettamente da questi valori.

Il risultato ottenuto nel numero di task completati, sebbene non ottimo come quello appena descritto, può essere considerato buono, considerando che coinvolge una moltitudine di parametri, tra cui il rapporto fra task *terminanti* e *uccisi*, la durata media dei task dei due gruppi e il numero di task in arrivo.

Il risultato peggiore ottenuto è quello riguardante il numero di task *evicted*, risultato essere circa due volte quello delle tracce reali. Dopo un'accurata analisi delle tracce però, si è notato che la media ottenuta dalle tracce reali non è un valore del tutto indicativo del reale andamento, perché avente una deviazione standard molto elevata (il valore in realtà si può osservare che

	Reale	Simulazione	Dev.Standard Reale	Diff Reale e Sim.
Running	124217	136037	11723	9%
Ready	5987	5726	6331	4%
Completed	3277	2317	2696	29%
Evict	1057	2165	2389	104%

Tabella 4.6: Risultati della simulazione con distribuzioni sintetizzate

oscilla in lunghi periodi sia su valori molto minori che molto maggiori a quello della simulazione). Il risultato può essere quindi spiegato considerando alcuni fattori:

- la simulazione opera su un valore medio di task in arrivo, sui dati reali invece la quantità di task in arrivo oscilla da valori elevati a valori bassi su lunghi periodi;
- la quantità di processi *evicted* dipende dalle politiche di scheduling e non è detto che cambi linearmente in base alla quantità di processi in *ready queue*;

Per questi motivi il risultato è stato considerato essere compatibile coi valori reali, loro stessi molto lontani dalla media a causa del carico oscillante in lunghi periodi.

Un altro fatto che potrebbe essere causa del diverso risultato nel numero di task *evicted* è il seguente: per quanto una singola distribuzione possa essere stata approssimata correttamente, in questo lavoro non sono state considerate le correlazioni fra diverse distribuzioni. Questo tipo di analisi richiede una quantità di lavoro ben più ampia di quella svolta e potrebbe essere una possibile estensione futura di questo progetto. In letteratura qualcosa in questo ambito è stato svolto, ma attualmente l'unico risultato ottenuto è stato trovare una correlazione fra le quantità di CPU e RAM richieste dai task, notando che all'aumentare della prima aumenta anche la seconda.

Conclusioni e sviluppi futuri

In questo lavoro di tesi è stato realizzato un modello simulato di un datacenter. Inizialmente sono state analizzate tracce rilasciate da Google, in modo da carpire più informazioni possibili sul funzionamento del loro datacenter, successivamente ne è stato implementato un modello e lo si è testato sulle tracce disponibili. È stata mostrata una similitudine fra lo stato reale delle tracce e lo stato ottenuto dalla simulazione, dando prova della correttezza delle semplificazioni effettuate e delle informazioni carpite. Successivamente si è cercato di sintetizzare le tracce, ottenendo buoni risultati e fornendo una strada per possibili miglioramenti.

Questo lavoro si inserisce in un progetto più ampio, che ha l'obiettivo di analizzare il funzionamento di un datacenter mediante tecniche provenienti dal *machine learning*, cercando di capire se ci si trova in stati pericolosi. Per effettuare questo lavoro sono necessarie tracce di datacenter in cui sono presenti sia informazioni riguardanti il normale funzionamento, sia informazioni riguardanti i guasti. Non essendo presenti tracce di questo tipo è stato svolto questo lavoro di tesi, che andrà esteso in futuro implementando un modello di guasti, volto a generare nuove tracce, comprendenti sia dati reali, sia guasti sintetizzati.

Un possibile modello di guasti potrebbe consistere nel porre in relazione fra loro tutte le entità della simulazione, creando un grafo orientato in cui ogni arco rappresenta la probabilità di propagazione di un guasto, o in modo complementare un arco potrebbe rappresentare ciò di cui un'entità ha bisogno per poter funzionare. Aggiungendo poi la probabilità di guasto di ogni entità

nel tempo si potrebbe quindi osservare in che modo ogni entità, guastandosi, può incidere sull'intero datacenter.

Appendice A

Generazione delle tracce

In questa sezione viene mostrato come creare le tracce fornibili al simulatore. Verrà mostrato come scaricare le tracce originali e come convertirle nel formato richiesto.

A.1 Download delle tracce originali

Le tracce sono salvate sul *Google Cloud Storage* e per ottenerle è necessario utilizzare GSUtil, ottenibile al seguente indirizzo:

`https://developers.google.com/storage/docs/gsutil_install?hl=it`

Per scaricare le tracce si può utilizzare il seguente comando:

```
gsutil cp -c -n -R gs://clusterdata-2011-1/ google-traces
```

Si noti che la dimensione totale dei file compressi è di circa 40 GB e che il download può richiedere molto tempo. Oltre a ciò, alcuni file risultano temporaneamente non disponibili, ed è quindi necessario ripetere più volte il comando affinché l'intero dataset venga scaricato.

Ottenute le tracce compresse si può notare che ognuna di esse risulta essere spezzata in vari frammenti, ognuno di essi compressi separatamente. Per ottenere tracce utilizzabili per le fasi future è necessario decomprimere

tutti i file ed effettuare un *merge* del risultato. È possibile effettuare ciò mediante i comandi (da eseguire per ogni traccia):

```
$ for a in *.gz; do gunzip $a; done
$ cat *.csv > name_of_trace.csv
```

Si noti che la dimensione totale dei file risultanti è di circa 200 GB.

A.2 Generazione delle tracce del simulatore

Il simulatore, come precedentemente descritto, utilizza 4 differenti tracce. Sono presenti 4 *script* che eseguono automaticamente tutti i passi necessari per generarle. Per permetterne il funzionamento è necessario caricare a priori i file delle tracce all'interno di un database SQLite, oltre che eseguire alcune query specificate nella documentazione.

Bibliografia

[1] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. 5 edition, 2011.

[2] John Wilkes. Omega, cluster management at google. Faculty Summit, 2011.

[3] Cade Metz. Return of the borg: How twitter rebuilt google's secret weapon. `http://www.wired.com/wiredenterprise/2013/03/google-borg-twitter-mesos/`, May 2013.

[4] Dave Lester. `https://speakerdeck.com/dlester/apache-mesos/`, October 2013.

[5] Cade Metz. Meet the data brains behind the rise of facebook. `http://www.wired.com/wiredenterprise/2013/02/facebook-data-team/`, February 2013.

[6] What is puppet? `http://puppetlabs.com/puppet/what-is-puppet`.

[7] Summary of the december 24, 2012 amazon elb service event in the us-east region. `http://aws.amazon.com/message/680587/`.

[8] Ben Treynor. Today's outage for several google services. `http://googleblog.blogspot.it/2014/01/todays-outage-for-several-google.html`, January 2014.

[9] The power failure seen around the world. `http://perspectives.mvdirona.com/2013/02/04/ThePowerFailureSeenAroundTheWorld.aspx`, February 2013.

[10] Alberto Montresor and Ozalp Babaoglu. The bison project. `http://www.cs.unibo.it/bison/publications/MB02.pdf`, 2002.

[11] Computer failure data repository. `https://www.usenix.org/cfdr`.

[12] John Wilkes. More Google cluster data. Google research blog, November 2011. Posted at `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`.

[13] Zitao Liu and Sangyeun Cho. Characterizing machines and workloads on a Google cluster. In *8th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems (SRMPDS)*, Pittsburgh, PA, USA, September 2012.

[14] Sheng Di, Derrick Kondo, and Walfredo Cirne. Characterization and comparison of cloud versus Grid workloads. In *International Conference on Cluster Computing (IEEE CLUSTER)*, pages 230–238, Beijing, China, September 2012.

[15] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Towards understanding heterogeneous clouds at scale: Google trace analysis. Technical Report ISTC–CC–TR–12–101, Intel science and technology center for cloud computing, Carnegie Mellon University, Pittsburgh, PA, USA, April 2012. Posted at `http://www.istc-cc.cmu.edu/publications/papers/2012/ISTC-CC-TR-12-101.pdf`.

[16] Guanying Wang, Ali R. Butt, Henry Monti, and Karan Gupta. Towards synthesizing realistic workload traces for studying the Hadoop ecosystem. In *19th IEEE Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*

(MASCOTS), pages 400–408, Raffles Hotel, Singapore, July 2011. IEEE Computer Society.

[17] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *ACM Symposium on Cloud Computing (SoCC)*, San Jose, CA, USA, October 2012.

[18] Asit K. Mishra, Joseph L. Hellerstein, Walfredo Cirne, and Chita R. Das. Towards cloud backend workloads: insights from Google compute clusters. *SIGMETRICS Perform. Eval. Rev.*, 37(4):34–41, March 2010.

[19] Bikash Sharma, Victor Chudnovsky, Joseph L. Hellerstein, Rasekh Rifaat, and Chita R. Das. Modeling and synthesizing task placement constraints in Google compute clusters. In *2nd ACM Symposium on Cloud Computing (SoCC)*, pages 3:1–3:14, Cascais, Portugal, October 2011. ACM.

[20] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Obfuscatory obscanturism: making workload traces of commercially-sensitive systems safe to release. In *3rd International Workshop on Cloud Management (CLOUDMAN)*, pages 1279–1286, Maui, HI, USA, April 2012. IEEE.

[21] M. Amoretti, A.L. Lafuente, and S. Sebastio. A cooperative approach for distributed task execution in autonomic clouds. In *21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 274–281, Belfast, UK, February 2013. IEEE.

[22] Ahmed Ali-Eldin, Maria Kihl, Johan Tordsson, and Erik Elmroth. Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. In *3rd Workshop on Scientific Cloud Computing (ScienceCloud)*, pages 31–40, Delft, The Nederlands, June 2012. ACM.

[23] Sheng Di, Derrick Kondo, and Walfredo Cirne. Host load prediction in a Google compute cloud with a Bayesian model. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, pages 21:1–21:11, Salt Lake City, UT, USA, November 2012. IEEE Computer Society Press.

[24] Sheng Di, Derrick Kondo, and Walfredo Cirne. Characterization and comparison of Google cloud load versus Grids. Technical Report hal-00705858 version 1, MESCAL (INRIA Grenoble Rhône-Alpes / LIG laboratoire d'Informatique de Grenoble), Grenoble, France, June 2012. Posted at `http://hal.archives-ouvertes.fr/hal-00705858`.

[25] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *SIGOPS European Conference on Computer Systems (EuroSys)*, pages 351–364, Prague, Czech Republic, 2013.

[26] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, November 2011. Revised 2012.03.20. Posted at URL `http://code.google.com/p/googleclusterdata/wiki/TraceVersion2`.

[27] Omnet++ simulation library: Continuous distributions. `http://www.omnetpp.org/doc/omnetpp/api/group__RandomNumbersCont.html`.

Ringraziamenti

Ringrazio chi mi ha supportato durante il percorso di questa laurea magistrale. Ringrazio il relatore, professor Ozalp Babaoglu, e il correlatore, Moreno Marzolla, che mi hanno seguito nello svolgimento di questa tesi.