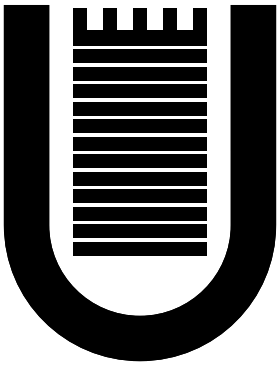
Università degli Studi di Roma “Tor Vergata”



Facoltà di Ingegneria

Laurea Triennale in Ingegneria Informatica

Posizionamento Ottimo per Applicazioni di Data Stream Processing

Relatore

Francesco Lo Presti

Candidato

Giacomo Marciani

Anno Accademico 2014/2015

Dedicato a Laura,
che mi ha dato il coraggio di credere in me stesso.

We should all be concerned about the future because
we will have to spend the rest of our lives there.

Charles F. Kettering, American inventor and businessman

Abstract

Da sempre l'essere umano acquisisce dati dal mondo esterno per estrarvi informazioni utili ad un processo decisionale o per un generale ampliamento della conoscenza.

Negli ultimi anni si assiste ad una proliferazione di dati mai vista prima d'ora. I volumi, il tasso di produzione e l'eterogeneità sono così elevati che ad oggi siamo in grado di generare più dati di quanto non sia possibile analizzarne. Tali tendenze sono destinate a crescere e diffondersi sempre più rapidamente. Ciò è dovuto alla naturale sinergia tra l'evoluzione delle tecnologie abilitanti, la propensione all'interazione digitale, e l'affermazione sul mercato di compagnie, beni e servizi dal core business data-oriented. Questa sinergia ha distribuito e potenziato la capacità di produzione e acquisizione di dati, al punto di mettere in ginocchio i tradizionali sistemi di processamento.

Questo fenomeno prende il nome di «Big Data» e, come vedremo, rende necessario lo sviluppo di tecnologie basate su paradigmi computazionali nuovi, orientati all'elaborazione tempestiva di flussi continui di dati. All'aumentare della frequenza di produzione dei dati, diminuisce la finestra temporale entro la quale uno specifico dato può assumere valore informativo. Un numero crescente di applicazioni richiede dunque un processamento sempre più sincrono di flussi continui di dati prodotti da sorgenti distribuite ed eterogenee, diretti al centro logico del sistema, ovvero laddove l'informazione viene resa fruibile.

Di fronte ad uno scenario così complesso, l'ottimizzazione dei processi di estrazione del valore informativo dei dati non è solo un affascinante esercizio accademico, bensì una vera e propria scelta di business, finanche una prerogativa al successo.

In questo lavoro di tesi affrontiamo un problema di ottimizzazione assai rilevante nel campo del Data Stream Processing, ovvero il posizionamento di un'applicazione sui nodi di un'architettura distribuita. Il nostro obiettivo è la ricerca di una formulazione del modello di ottimizzazione quanto più possibile efficiente, di modo che possa risultare un utile bechmark per lo sviluppo di soluzioni euristiche al problema.

Forniamo dapprima una panoramica sui Big Data, spiegando come tale fenomeno renda necessario lo sviluppo di nuovi paradigmi computazionali per il processamento dei dati, e mostrando come tali paradigmi si allontanino dalle soluzioni tradizionali. Introduciamo dunque il concetto di sistema di Information Flow Processing, spiegando come in esso possano essere riconosciuti i sistemi di Data Stream Processing e di Complex Event Processing. Tale panoramica ci armerà degli strumenti necessari a comprendere la natura del problema in esame.

Il problema del posizionamento per applicazioni di Data Stream Processing consiste nell'assegnare l'esecuzione delle unità procedurali di un'applicazione DSP ai nodi computazionali di un'architettura distribuita. Quando il processamento di un flusso continuo di dati è distribuito su risorse computazionali e trasmissive distinte, determinare una soluzione di posizionamento quanto più possibile prossima all'ottimalità è di fondamentale importanza per garantire l'efficienza e l'affidabilità dell'applicazione.

Proponiamo tre formulazioni in programmazione lineare binaria per il problema di posizionamento. Partendo da una prima formulazione di base, abbiamo potuto ricavare le seguenti apportando opportune modifiche al dominio delle variabili decisionali e ad uno specifico vincolo di ammissibilità.

I modelli di ottimizzazione proposti sono stati dapprima implementati in OPL, all'interno dell'ambiente di sviluppo integrato IBM CPlex Optimization Studio. Tale approccio ci ha garantito una prototipazione rapida dei modelli, al fine di verificarne quanto prima consistenza ed efficienza su piccole istanze ad hoc. Questi modelli sono stati poi implementati in Java, costituendo così il progetto OPMap, il quale utilizza la libreria IBM CPlex Concert Technology per la loro compilazione e risoluzione. L'implementazione in Java ci ha permesso il totale controllo delle istanze del problema, così da poter verificare l'efficienza dei modelli quando applicati a scenari più complessi e realistici.

I modelli sono stati successivamente sottoposti ad un'analisi sperimentale quanto più possibile estensiva, volta a compararne le prestazioni in funzione della dimensione e distribuzione dell'istanza.

I risultati ottenuti sono molto positivi, in quanto dimostrano il netto miglioramento prestazionale della formulazione finale rispetto a quella di partenza.

Ringraziamenti

Questa tesi porta con sè anni di emozioni. Desidero ringraziare tutte quelle persone che queste emozioni le hanno suscitate o vi hanno preso parte.

Ringrazio il professor Lo Presti per avermi posto, più volte, dinnanzi a sfide entusiasmanti. Ringrazio Matteo Nardelli per avermi accompagnato in questo lavoro di tesi: non avrei potuto desiderare supervisor migliore. Ringrazio il professor Tucci per avermi sempre dato il consiglio giusto, al momento perfetto. Ringrazio il professor Pettorossi per avermi trasmesso passione per lo studio, e per essermi stato accanto nei momenti difficili.

Ringrazio la mia famiglia per aver sostenuto emotivamente ed economicamente il mio percorso in questi anni. In modo particolare ringrazio mio padre per avermi sempre fatto sentire importante, per aver valorizzato i miei interessi e stimolato la mia curiosità. Ringrazio mio nonno per aver gettato il seme della mia passione per l'informatica: vorrei che oggi fosse qui a vederne un primo risultato.

Ringrazio i miei amici di sempre, Michele Finocchi, Francesco Sposato, Giuliano Bartoloni ed Edoardo Croce per avermi regalato la spensieratezza anche quando non ero capace di concedermela. In particolar modo ringrazio Michele e tutta la sua famiglia, per avermi sempre fatto sentire a casa. Ringrazio Daniele Oteri perchè ogni volta che ci rivediamo mi si spalanca il cuore. Ringrazio Gabriele Santi perchè anche nelle giornate più intense riusciamo a darci la carica. Ringrazio Edoardo Felici perchè ogni estate mi convince a vedere al di là del mio computer.

Ringrazio Laura perchè senza di lei oggi non sorriderei. La ringrazio per non aver mai dubitato di me in questi anni, anche quando ero io il primo a farlo. La ringrazio per avermi sempre dato il coraggio di superare i miei limiti. La ringrazio perchè con lei tutto sembra realizzabile.

Indice

Capitolo 1

Data Stream Processing

Il Data Stream Processing è un paradigma computazionale che modella una elaborazione come un processamento modulare e attivo di un flusso continuo di dati. Esso si oppone ai paradigmi tradizionali che considerano l'elaborazione come un processo discreto e passivo.

Per comprendere a fondo un'innovazione è necessario conoscere i presupposti tecnologici e culturali che ne hanno motivato lo sviluppo, che ne motivano l'esistenza e ne consolideranno l'utilizzo. Nel caso del Data Stream Processing, il presupposto principale risiede nel fenomeno «Big Data».

In questo capitolo daremo una definizione di big data, spiegando cosa distingua un big dataset da un dataset tradizionale, e per quale motivo il fenomeno abbia messo in crisi i tradizionali sistemi di elaborazione. Daremo poi una definizione di Data Stream Processing, contestualizzando il paradigma all'interno dell'Information Flow Processing, e spiegando ciò che lo distingue dal Complex Event Processing. Infine daremo una prima rappresentazione di applicazione di data stream processing, che ponga le basi per la trattazione del problema di posizionamento degli operatori.

1.1 Big Data

Con il termine «Big Data» si identifica un fenomeno di generazione distribuita massiccia e rapida di dati digitali eterogenei e dall'elevato potenziale informativo [1]. Tale fenomeno sfugge al controllo dei tradizionali sistemi

di elaborazione, in quanto impedisce un approccio che preveda una fase di memorizzazione preliminare al processamento. Nel seguito chiameremo «big dataset» un dataset conforme alle peculiarità del fenomeno.

Trend produzione dati Ogni giorno produciamo 2.5 quintilioni di bytes di dati, ed il 90% dei dati presenti nel mondo oggi è stato creato solo negli ultimi due anni [2]. Nel 1992 il traffico Internet globale ammontava a 100GB al giorno, nel 2013 si sono raggiunti i 28.875GB al secondo e nel 2018 sono previsti 50.000GB al secondo.

Solo il 10% dei dati prodotti viene attualmente analizzato. Considerando la bassa densità di valore dei Big Data, stiamo perdendo molto più del 90% del loro valore. Uno studio della IDC stima che nel 2020 il 33% dell'universo digitale conterrà dati ad alto potenziale informativo [3]. Essendo l'80% dei dati prodotti non strutturati, non sono di facile analisi.

Il traffico dati mobile è cresciuto del 55% nell'ultimo anno, con un aumento del 15% solo dal primo al secondo quarto del 2015[4]

Bisogna resistere alla impulsiva diffidenza per gli studi prospettici: le stime elaborate sono state sempre smentite da valori molto più elevati di quelli previsti. Lo stesso studio pubblicato dalla IDC nel 2007 [5], prevedeva una produzione nel 2010 di 988 EB. I dati attuali mostrano una produzione di 1227 EB, pari al 29.19% in più.

Valore dei Big Data Dal 2005 ad oggi l'interesse del mondo accademico ed industriale per i Big Data è cresciuto vertiginosamente. Si conta un aumento del numero di papers pubblicati del 1000%[6], ed un aumento delle ricerche su Google[1] del 4900% [7]. Il crescente interesse per i Big Data non può imputarsi solo all'evidenza del fenomeno, bensì anche all'immenso valore economico derivante dallo sviluppo di tecnologie capaci di estrarvi valore informativo. Il valore di un dataset è in generale solo potenziale: è la capacità di analizzarlo inferendovi informazione utile a conferirvi valore. Lo spettro del valore informativo è certamente ampliato dall'eterogeneità delle sorgenti dati, ma poichè tale valore ha una scarsa densità, il rischio di perderlo è un problema certamente attuale.

[1] trend relativo alle ricerche riguardanti i Big Data nel settore industriale, ovvero hardware, software e servizi.

Lo studio di forecasting 2016-2026 sul mercato Big Data condotto da Wikibon mostra un trend di crescita tipico delle tecnologie disruptive[2] [9].

Mercato Big Data Il numero di compagnie costruite o coadiuvate da un business model data-driven è in continua crescita [10, 11]. Tutti oggigiorno, dalle grandi compagnie alle piccole startup, possono archiviare e trasferire a basso costo grandi quantità di dati. Basti pensare che dal 2005 ad oggi il costo di archiviazione[3] si è ridotto del 63.81% [12], mentre il traffico dati mobile è cresciuto del 55% solo nell'ultimo anno [4]. Le tecnologie abilitanti dunque certamente non mancano. La sfida però non si pone tanto sull'archiviazione o sulla trasmissione dei dati, divenute ormai prerogative, bensì sulla loro analisi e sulla capacità di estrarvi valore.

Uno studio recente condotto dalla IDC mostra una crescita del mercato Big Data[4] caratterizzata da un CAGR[5] del 24.6%, raggiungendo un valore di 41.5 miliardi di dollari entro il 2019 [14].

1.1.1 Metriche Big Data

Il fenomeno Big Data non deve sfuggire ad una rappresentazione quantitativa. E' necessario definire uno spazio metrico non solo per poter distinguere un big dataset da un dataset tradizionale, ma anche per poter valutare le soluzioni tecnologiche sviluppate per il loro processamento.

In letteratura sono stati proposti due spazi metrici alternativi, attualmente accettati a livello accademico ed industriale. Questi spazi sono applicabili ad un dataset in generale. Diciamo dunque che un dataset è un big dataset nel momento in cui presenta alti valori in ognuna delle metriche dello spazio. Ne consegue che la definizione di big dataset è relativa alle tecno-

[2] un'innovazione è definita «disruptive» quando è in grado di aprire un nuovo mercato ed influenzare fortemente i mercati preesistenti. Esempi di innovazioni disruptive sono l'invenzione del telefono e del personal computer [8].

[3] stiamo assumendo qui che il costo di archiviazione possa intendersi in $/GB. Questa è una approssimazione lecita, ma superficiale, considerando l'eterogeneità degli odierni sistemi e sservizi di storage.

[4] mercato da intendersi come aggregato dei settori hardware, software e servizi.

[5] il Compound Annual Growth Rate (CAGR) è un indice composto di crescita su base annua, utilizzato nelle analisi prospettiche per fotografare l'appetibilità di un investimento [13].

logie di riferimento: il potenziamento tecnologico innalza il threshold della definizione[6].

Spazio metrico V^5 Lo spazio V^5, o «spazio delle cinque V», individua il dataset mediante le seguenti metriche:

Volume dimensione del dataset. Un big dataset è tipicamente misurato in multipli di terabytes e petabytes.

Velocità il datarate delle sorgenti che alimentano il dataset. Un big dataset è alimentato da sorgenti ad altissimo datarate.

Variabilità eterogeneità della natura dei dati che compongono il dataset e variabilità nel datarate delle sorgenti che lo alimentano. Un big dataset raccoglie dati di natura profondamente diversa, e li raccoglie da sorgenti che possono generare un traffico molto piccato.

Veracità inaffidabilità o mancanza di precisione delle sorgenti dati, e dunque inaffidabilità informativa dei dati prodotti.

Valore valore informativo del dato. I big data hanno un alto potenziale informativo, ma una basse densità di tale valore. Questo vuol dire che il valore effettivo di un big dataset viene estrapolato solo identificando dei pattern su tutto il dataset, e non su una sua porzione ristretta.

Lo spazio metrico V^5 è attualmente il più diffuso in letteratura [1, 16]. Storicamente esso deriva dallo spazio V^3, o «spazio delle tre V», a cui è stata successivamente aggiunta la dimensione del valore e della veracità [17]. La mancanza del valore informativo come metrica big data, esponeva infatti lo spazio V^3 ad un simpatico paradosso che ha motivato la definizione dello spazio metrico C^3 [18].

[6] in uno studio condotto dalla IBM nel 2012, un dataset con una dimensione di un terabyte veniva già considerato big dataset. Oggi, un big dataset è tipicamente misurato in multipli di terabytes e petabytes [15].

Spazio metrico C^3 Lo spazio metrico C^3, o «spazio delle tre C», è stato introdotto per ovviare ad un paradosso a cui era esposto lo spazio V^3 [18]. Se consideriamo infatti un dataset composto da N record, dove l'i-esimo record è una sequenza di n ripetizioni del numero $i \in [0, N]$, crescente velocemente all'infinito, lo spazio metrico V^3 classificherebbe tale dataset come big dataset, nonostante tale considerazione sia intuitivamente falsa. Per ovviare a tale paradosso è stato introdotto lo spazio metrico C^3 , le cui dimensioni sono (i) la Cardinalità, ovvero la dimensione in record del dataset, (ii) la Continuità, ovvero la dimensione utile di ogni record, e (iii) la Complessità, ovvero la domanda computazionale del processamento. L'introduzione della complessità come metrica big data risolve il paradosso dello spazio V^3. Nonostante ciò, lo spazio V^5 è rimasto comunnque il più adottato.

1.2 Information Flow Processing

Con il termine «Information Flow Processing» (IFP) ci si riferisce ad un dominio applicativo in cui l'utente deve tempestivamente estrarre valore da un massiccio flusso continuo di dati eterogenei prodotti da sorgenti distribuite [19]. Questi dati fluiscono dalla periferia del sistema fino al suo centro[7], essendo processati «on the fly» nel momento stesso in cui vi transitano. Tale dominio ha uno spettro di applicabilità oggigiorno molto ampio[8].

Sistemi tradizionali I DBMS tradizionali prevedono (i) una fase di memorizzazione[9] ed indicizzazione dei dati prima di poterli processare, e (ii) che l'elaborazione dei dati venga esplicitamente richiesta dall'utente. Tali sistemi sono pertanto chiamati Human-Active Database-Passive (HADP), per indicarne l'interazione passiva ed asincrona rispetto all'arrivo dei dati nel sistema. Nei sistemi tradizionali includiamo anche gli Active-DBMS, sviluppati per garantire una certa reattività all'arrivo dei dati. Questi sistemi permettono di definire delle procedure da eseguire al verificarsi di specifiche condizioni. Ciò realizza, almeno in teoria, un'operatività sin-

[7] «periferia» e «centro» del sistema sono intesi in senso logico, non solo geografico, per quanto le due accezioni possano spesso coincidere.

[8] la sentiment analysis, la finanza, l'high frequency trading, il monitoraggio delle reti, la sicurezza ed il controllo industriale sono solo alcuni classici esempi della sua applicabilità.

[9] intesa come memorizzazione persistente.

crona all'arrivo dei dati al sistema. Tuttavia, tali sistemi non scalano, nè computazionalmente nè economicamente, quando applicati al dominio IFP.

I DBMS tradizionali non soddisfano dunque i requisiti previsti dall'IFP, aprendo così le porte allo sviluppo di sistemi basati su paradigmi computazionali totalmente differenti. Negli anni sono emersi, dalla comunità scientifica ed industriale, due modelli principali: il Data Stream Processing ed il Complex Event Processing.

Data Stream Processing Il Data Stream Processing (DSP) è un paradigma computazionale in cui una rete di operatori processa un flusso continuo di tuple. Ogni operatore incapsula la logica di un'azione da eseguire sul flusso in ingresso, producendo così un conseguente flusso in uscita. L'insieme degli operatori è partizionato in (i) sources, ovvero le sorgenti prime del flusso, (ii) sinks, ovvero i destinatari ultimi del flusso, e (iii) pipes[10], ovvero coloro che attuano trasformazioni o individuazione di patterns sul flusso entrante riportadone il risultato sul flusso in uscita. La natura di queste azioni dipende evidentemente dalla semantica del linguaggio adottato dal sistema.

Complex Event Processing Il Complex Event Processing (CEP) è un paradigma computazionale in cui l'oggetto del processamento è un flusso continuo di eventi[11], sui quali è possibile definire complessi patterns relazionali. L'origine di tali sistemi si fa risalire al dominio applicativo Publish-Subscribe (PS) [20]. Un sistema PS notifica infatti singoli eventi agli agenti che vi abbiano espresso interesse[12]. Ciò che differenzia profondamente i PS dai CEP è il fatto che i primi elaborano e notificano singoli eventi, mentre i secondi estendono l'elaborazione e la notifica a complessi patterns definiti sugli stessi, disponendo dunque di un linguaggio di sottoscrizione molto più espressivo.

Framework IFP Le tecnologie del dominio IFP sono il frutto sinergico di diverse comunità scientifiche. Non è dunque semplice fornire definizioni

[10] questi operatori assumono nomi diversi in letteratura: pipes, cosumers, bolts, e così via.

[11] per evento si intende il verificarsi di una certa condizione sui dati

[12] l'interesse di un agente può essere espresso (i) in base al topic dell'evento, ovvero la sua appartenenza ad una specifica categoria, oppure (ii) in base al suo contenuto. Si parla rispettivamente di sistemi topic-based e sistemi cotnent-base.

e metriche stabili. In [19] si è cercato di raccogliere tutti questi contributi, definendo un framework di modellazione che permette di identificare dettagliatamente le specifiche di un sistema IFP. Tale framework prende in esame (i) il tipo di processamento, (ii) le modalità della sua dichiarazione, (iii) l'interazione tra gli operatori ed il sistema, (iv) la natura dei dati processati, (v) la percezione del tempo, e (vi) la semantica del linguaggio adottato.

1.3 Applicazioni di Data Stream Processing

Un'applicazione di data stream processing può essere rappresentata come un grafo orientato aciclico, dove i nodi rappresentano gli operatori e gli archi rappresentano i data stream. In Figura 1.1 ne riportiamo una rappresentazione generale, mentre il Figura 1.2 ne riportiamo una presa da un caso reale [21]. Nel Capitolo 2 daremo una caratterizzazione dettagliata dei nodi e degli archi del grafo in 1.1, al fine di renderlo strumento espressivo per la trattazione del problema del posizionameto degli operatori.

Architettura e posizionamento Un'applicazione di data stream processing è pensata per essere eseguita da una architettura distribuita, sia essa un Cluster, una Cloud, una Grid, una Fog o una soluzione ibrida.

Il problema di determinare il deploy ottimale dell'applicazione sui nodi dell'architettura è comunemente chiamato Operator Placement Problem (OPP)[13] e verrà largamente affrontato a partire dal prossimo capitolo. La scelta del tipo di infrastruttura è strettamente legata al problema di posizionamento, ma esula lo scopo di questa tesi. Nei capitoli seguenti descriveremo dunque l'architettura distribuita nella sua accezione più generale, ovvero come un grafo orientato pesato e completo.

Ottimizzazioni Una applicazione DSP può essere ottimizzata apportando modifiche strategiche alla sua topologia o al posizionamento degli operatori [22]. Tali modifiche possono intendersi sia come soluzioni progettuali, che come ottimizzazioni adattative da eseguire a runtime. Le principali sono (i) il riordinamento degli operatori, (ii) l'eliminazione della ridondanza, (iii)

[13] anche detto Operator Deployment Problem (ODP).

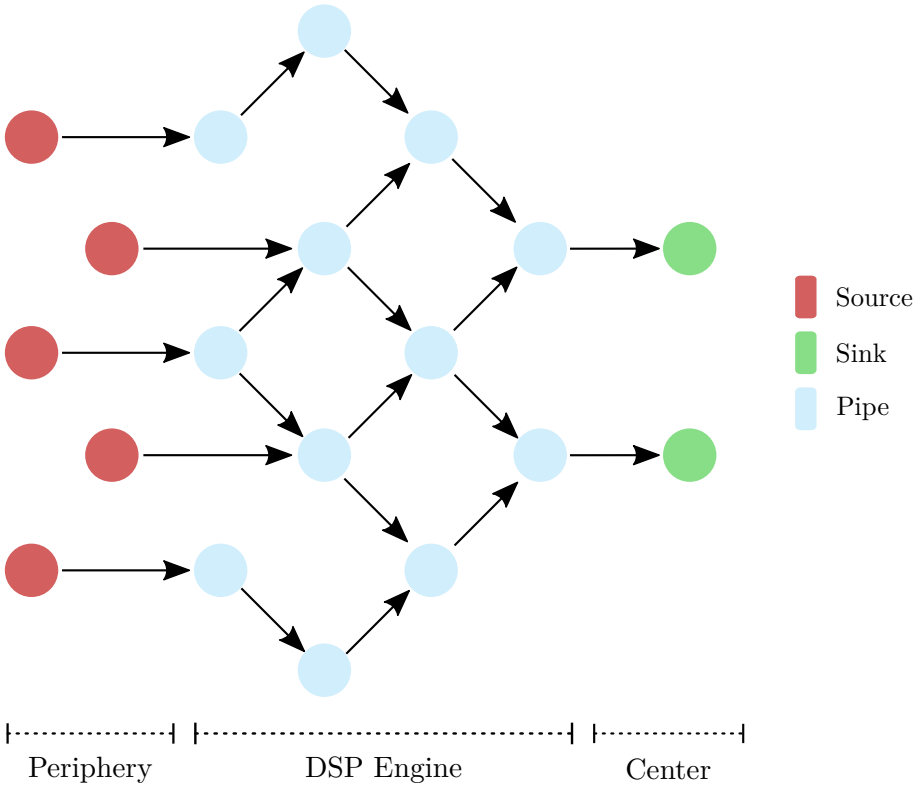


Figura 1.1: Rappresentazione generale di un'applicazione di data stream processing.

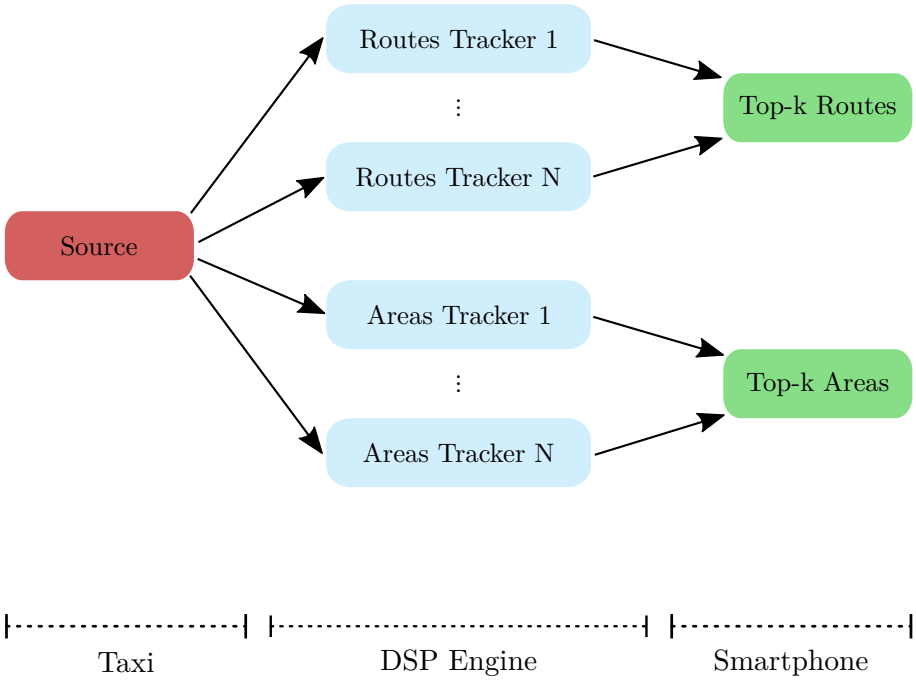


Figura 1.2: Rappresentazione di un'applicazione reale di data stream processing [21].

la fusione degli operatori, (iv) la fissione degli operatori, (iv) la condivisione dello stato fra operatori, (v) la selezione degli operatori, e (vi) il load shedding. Alcune di esse apportano modifiche locali al numero dei nodi dell'applicazione. E' dunque necessario considerare questa variabilità, nel design di modelli ed algoritmi di ottimizzazione. Nell'analisi sperimentale condotta nel Capitolo 4, terremo conto di questa variabilità considerando in fase di test un numero di nodi operazionali maggiore di quelli tipicamente previsti nel design dell'applicazione [21].

Esempi di applicazioni Spesso si associa il fenomeno Big Data (e tecnologie annesse) all'esplosione dei social media. Questo perchè l'analisi dei comportamenti digitali dell'utente-cosumatore ha una grande importanza strategica ed economica nei processi di business intelligence. Risulta dunque abbastanza intuitivo pensare ad applicazioni come Google Hot Searches [23], o a strumenti di sviluppo come Twitter Streaming API [24]. Il fenomeno Big Data però non è da confinarsi ai social media. La sua applicabilità si estende al monitoraggio delle reti, all'analisi del consumo energetico, al monitoraggio

dei mercati finanziari, all'high frequency trading ed altro ancora.

Un esempio lampante di applicazione data stream processing nel mondo della ricerca scientifica è la selezione dei dati generati dal Large Hadron Collider (LHC) del CERN di Ginevra [25]. Ogni esperimento condotto sul LHC implica l'osservazione di 600 milioni di collissioni atomiche al secondo, di cui tipicamente solo lo 1% ha una rilevanza scientifica. Queste collisisoni producono un dataset ad un rate medio di 1Pbps, il quale viene filtrato in realtime dal datacenter per selezionarne la sola porzione interessante su cui poi condurre analisi nei centri specializzati distribuiti in tutto il pianeta [26, 27].

Benchmarks Lo studio di soluzioni basate su nuovi paradigmi computazionali rende necessaria la definizione di un benchmark che ne orienti lo sviluppo. Nel caso del data stream processing, ciò è stato realizzato dalla Association of Computing Machinery (ACM) attraverso la conferenza Distributed Event Based Systems (DEBS).

La Distributed Event Based Systems (DEBS) Grand Challenge è una competizione internazionale affermatasi, a partire dal 2012, come benchmark della applicabilità di soluzioni IFP [28, 29, 30, 31]. Ogni anno il DEBS Grand Challenge propone un nuovo problema considerato rilevante per il mondo della ricerca e dell'industria. Di tale problema fornisce una descrizione dettagliata, un dataset realistico e un insieme di metriche volte a quantificare la scalabilità delle soluzioni proposte[14]. I partecipanti al concorso sottomettono dunque al giudizio della commissione soluzioni innovative e fortemente performanti. Queste soluzioni permettono di tracciare annualmente lo stato dell'arte del data stream processing.

[14] ad esempio: query throughput e query latency in funzione del workload e del numero di nodi computazionali.

Capitolo 2

Posizionamento degli operatori

Il problema del posizionamento degli operatori consiste nell'assegnare l'esecuzione degli operatori di una applicazione DSP ai nodi computazionali di una architettura distribuita. Quando il processamento di una standing query è distribuito su risorse computazionali e trasmissive distinte, determinare una soluzione di posizionamento quanto più possibile prossima all'ottimalità è di fondamentale importanza per garantire l'efficienza dell'applicazione. Il posizionamento degli operatori può dunque considerarsi una tecnica di ottimizzazione per un'applicazione di data stream processing [22].

Una applicazione DSP è rappresentata da una topologia che incapsula le relazioni fra gli operatori e i dettagli relativi alla loro domanda computazionale. Una architettura distribuita, sia essa una Cloud o una Grid, è rappresentata da una topologia che incapsula le potenzialità operative e la connettività delle risorse computazionali. Il problema del posizionamento degli operatori consiste dunque nel mappare la topologia operazionale della applicazione sulla topologia logica e fisica delle risorse computazionali offerte dalla architettura distribuita. Il posizionamento degli operatori è un problema di ottimizzazione combinatoria riducibile al Binary Knapsack Problem ed al Bin Packing Problem: pertanto esso appartiene alla classe di complessità computazionale NP-hard [32, 33, 34]. Risulta dunque corretto, oltre che intuitivo, esprimerlo mediante un modello di programmazione lineare binaria.

In questo capitolo descriviamo nel dettaglio il modello adottato per risolvere il problema di posizionamento degli operatori. Talvolta faremo riferimento a problemi di ottimizzazione combinatoria, noti in letteratura: come il Knapsack Problem [35, 36, 37], il Bin Packing Problem [38, 39] ed il Metric Labeling Problem [40, 41]. Nella Sezione 2.1 descriviamo il grafo che permette di rappresentare una applicazione DSP. Nella Sezione 2.2 descriviamo il grafo che permette di rappresentare una architettura distribuita. Nella Sezione 2.3 definiamo la soluzione del problema del posizionamento degli operatori, il vettore di posizionamento dei data stream e le condizioni necessarie e sufficienti alla ammissibilità. Nella Sezione 2.4 descriviamo lo spazio metrico adottato nel nostro modello, formulando il tempo di risposta della applicazione e la disponibilità della architettura. Introduciamo inoltre le utili definizioni di metrica antitetica e metrica disomogenea. Nella Sezione 2.5 descriviamo il criterio di ottimizzazione adottato nel nostro modello, mostrando come questo derivi dall'approccio di Čebyšëv ai problemi di ottimizzazione multi-obiettivo. Nella Sezione 2.6 diamo infine la formulazione in programmazione lineare binaria del problema: di essa forniamo una versione basica ed una sua variante derivata dal vincolo di conservazione del flusso del Metric Labeling Problem.

2.1 Grafo DSP

Una applicazione DSP è rappresentata da un grafo diretto aciclico $G_{dsp} = (V_{dsp}, E_{dsp})$, dove V_{dsp} è l'insieme dei nodi operazionali e E_{dsp} è l'insieme dei data stream (Figura 2.1). Ogni nodo $i \in V_{dsp}$ incapsula la domanda computazionale dell'istanza dell'operatore op_i. Ogni arco $(i, j) \in E_{dsp}$ incapsula le caratteristiche del data stream che connette la istanza dell'operatore op_i alla istanza dell'operatore op_j. Qualora $(i, j) \in E_{dsp}$, diciamo che il nodo operazionale i «produce» il data stream (i, j), mentre il nodo j «consuma» il data stream (i, j), e ancora che un flusso di tuple è «associato» al data stream (i, j), ovvero che delle tuple «costituiscono» il data stream (i, j).

2.1.1 Nodi operazionali

Un generico nodo $i \in V_{dsp}$ è detto «nodo operazionale», in quanto ad esso è associata l'istanza di un operatore. Esso è caratterizzato da:

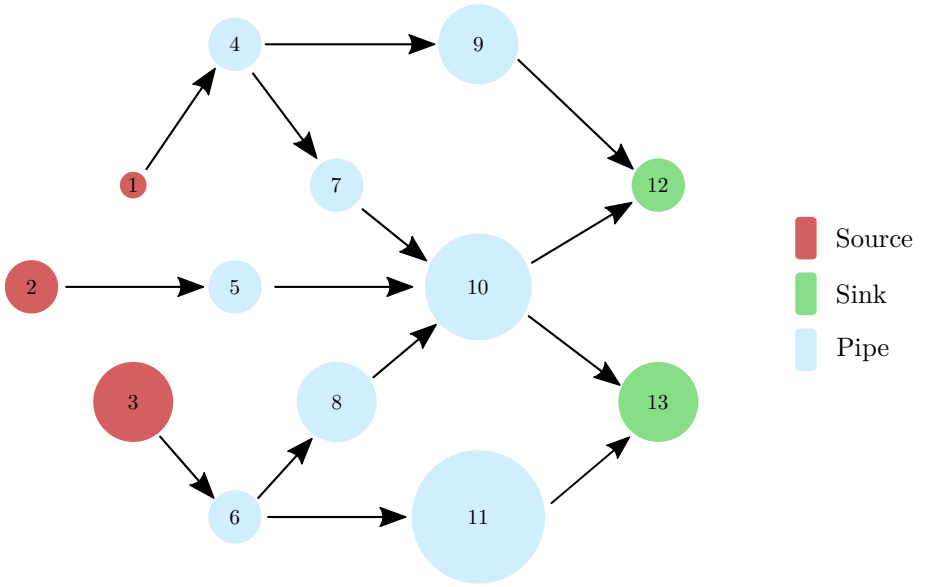


Figura 2.1: Esempio di Grafo DSP. Ogni nodo è etichettato con il proprio identificativo. La dimensione di un nodo i sta ad indicare la relativa domanda computazionale, in termini di risorse richieste $\mathbf{C}_i$ e velocità di processamento S_i . La dimensione degli archi (i, j) sta ad indicare la entità del data stream, in termini di flusso $\varphi_{(i,j)}$ e dimensione delle tuple $\lambda_{(i,j)}$. Per una maggiore chiarezza grafica, abbiamo assunto data stream omogenei. I nodi «Source» sono i nodi $i \in So$. I nodi «Sink» sono i nodi $i \in Si$. I nodi «Operator» sono i nodi $i \in V_{dsp} \setminus So \cap Si$. Se non diversamente specificato, assumiamo che $\forall i \in V_{dsp} : V^i_{res} = V_{res}$

op_i	Istanza dell'operatore assegnato al nodo.
$\mathbf{C}_i$	Vettore delle risorse necessarie all'esecuzione dell'operatore. Una componente $c_{i,\varrho} \in \mathbf{C}_i$ indica la domanda di risorsa ϱ per l'istanza dell'operatore.
S_i	Velocità di processamento delle tuple *da parte di un processore di riferimento*. Misurata in tuple al secondo ($[\tau/s]$).
φ_i^{in}	Flusso delle tuple in ingresso all'operatore. Misurato in tuple ($[\tau]$).
φ_i^{out}	Flusso delle tuple in uscita dall'operatore. Misurato in tuple ($[\tau]$).

λ_i^{in} Dimensione delle tuple in ingresso all'operatore. Misurata in bit per tupla tupla ($[b/\tau]$).

λ_i^{out} Dimensione delle tuple in uscita dall'operatore. Misurata in bit per tupla tupla ($[b/\tau]$).

$V_{res}^i \subseteq V_{res}$ Nodi computazionali $u \in V_{res}$ sui quali il nodo i può essere posizionato.

Nodi source e nodi sink Il grafo G_{dsp} presenta almeno un nodo source e almeno un nodo sink, per i quali alcuni specifici parametri topologici ed operativi sono vincolati ad opportuni valori. Denotiamo con So l'insieme dei nodi source, e con Si l'insieme dei nodi sink, dove $So, Si \subset V_{dsp}$.

Definizione 2.1. Un *nodo source* è un nodo operazionale che produce e non consuma un data stream. Esso è dunque un nodo $i \in V_{dsp}.op_i = src$ che ha almeno un arco uscente ($deg_{out}(i) \geq 1$), non ha alcun arco entrante ($deg_{in}(i) = 0$, $\varphi_i^{in} = 0$, $\lambda_i^{in} = 0$), e può essere posizionato esclusivamente su uno specifico nodo $u \in V_{res}$ ($|V_{res}^i| = 1$).

Tale definizione può essere rilassata considerando nodi source i per i quali $|V_{res}^i| \geq 1$.

Definizione 2.2. Un *nodo sink* è un nodo operazionale che consuma e non produce un data stream. Esso è dunque un nodo $i \in V_{dsp}.op_i = snk$ che ha almeno un arco entrante ($deg_{in}(i) \geq 1$), non ha alcun arco uscente ($deg_{out}(i) = 0$, $\varphi_i^{out} = 0$, $\lambda_i^{out} = 0$), e può essere posizionato esclusivamente su uno specifico nodo $u \in V_{res}$ ($|V_{res}^i| = 1$).

Tale definizione può essere rilassata considerando nodi sink i per i quali $|V_{res}^i| \geq 1$.

2.1.2 Data stream

Un generico arco $(i, j) \in E_{dsp}$ è detto «data stream», in quanto ad esso è associato un flusso di tuple. Esso è caratterizzato da:

$\varphi_{(i,j)}$ Flusso delle tuple in transito nel data stream dal nodo $i \in V_{dsp}$ al nodo $j \in V_{dsp}$. Misurato in tuple ($[\tau]$).

$\lambda_{(i,j)}$ Dimensione delle tuple in transito nel data stream dal nodo $i \in V_{dsp}$ al nodo $j \in V_{dsp}$. Misurato in bit per tupla ($[b/\tau]$).

Additività del data stream Un operatore può dover consumare data stream prodotti da operatori differenti. In tal caso i data stream prodotti vengono cumulati nell'operatore che li consuma. Vale dunque la seguente *proprietà di additività del data stream.*

Proposizione 2.1. *Ogni nodo operazionale può consumare data stream prodotti da operatori differenti, ovvero*

$$\varphi_j^{in} = \sum_{(i,j)\in E_{dsp}} \varphi_{(i,j)} \qquad \forall j \in V_{dsp} \tag{2.1}$$

dove φ_j^{in} è il flusso di tuple in ingresso al nodo operazionale j, e $\varphi_{(i,j)}$ è il flusso di tuple di ogni data stream diretto al nodo operazionale j.

Omogeneità delle tuple Qualora un operatore consumi data stream prodotti da operatori differenti, assumiamo per semplicità che la dimensione delle tuple consumate sia omogenea. Vale dunque la seguente *proprietà di omogeneità del data stream.*

Proposizione 2.2. *Ogni nodo operazionale può consumare solo tuple della stessa dimensione, ovvero*

$$\lambda_{(i,j)} = \lambda_j^{in} \qquad \forall (i,j) \in E_{dsp} \tag{2.2}$$

dove $\lambda_{(i,j)}$ è la dimensione delle tuple che costituiscono il data stream (i,j), ed λ_j^{in} è la dimensione delle tuple in ingresso al nodo operazionale j che da esso sono consumate.

Trasformazione del data stream In generale un operatore esegue una trasformazione del data stream. Tale trasformazione può dunque conservare o meno la entità del data stream, in termini di flusso e dimensione delle tuple. Ogni operatore op_i è caratterizzato da una funzione di trasformazione $T_i : \mathbb{R}^2 \to \mathbb{R}^2$ che associa flusso e dimensione delle tuple in ingresso al flusso e dimensione delle tuple in uscita. Vale dunque la seguente *proprietà di trasformazione del data stream.*

Proposizione 2.3. *Ogni nodo operazionale può non conservare il data stream, ovvero*

$$\varphi_i^{in} \neq \varphi_i^{out} \vee \lambda_i^{in} \neq \lambda_i^{out} \qquad \exists i \in V_{dsp} \tag{2.3}$$

dove φ_i^{in} [λ_i^{in}] è il flusso [la dimensione] delle tuple in ingresso al nodo operazionale j, e φ_i^{out} [λ_i^{out}] è il flusso [la dimensione] delle tuple in uscita dal nodo operazionale j.

Cammini In un grafo G_{dsp} siamo interessati a studiare il processo di trasformazione dei data stream, dalla loro prima produzione in un nodo source, al loro consumo finale in un nodo sink. Se non diversamente specificato, $\Pi_{G_{dsp}}$ indica l'insieme di tutti i cammini orientati sul grafo G_{dsp}, tali da connettere un nodo source ad un nodo sink. Un generico cammino $\pi \in \Pi_{G_{dsp}}$ di dimensione n_π è dunque una sequenza $\pi = (i_k \in V_{dsp})_{k \in [1,n_\pi]}$ di nodi appartenenti al cammino, in cui i_1 è un nodo source e i_{n_π} è un nodo sink.

2.2 Grafo RES

Una architettura distribuita è rappresentata da un grafo diretto completamente connesso $G_{res} = (V_{res}, E_{res})$, dove V_{res} è l'insieme dei nodi computazionali e E_{res} è l'insieme del link logici (Figura 2.2). Ogni nodo $u \in V_{res}$ incapsula la offerta computazionale della risorsa. Ogni arco $(u, v) \in E_{res}$ incapsula le potenzialità trasmissive del link logico che connette la risorsa $u \in V_{res}$ alla risorsa $v \in V_{res}$. Qualora $(u, v) \in E_{res}$, diciamo che il nodo u ha la possibilità di «inviare» un data stream al nodo v, mentre il nodo v ha la possibilità di «ricevere» un data stream dal nodo u.

2.2.1 Nodi computazionali

Un generico nodo $u \in V_{res}$ è detto «nodo computazionale», in quanto può eventualmente eseguire le computazioni richieste da un nodo operazionale. Esso è caratterizzato da:

$\mathbf{C}_u$ Vettore delle risorse offerte.

Una componente $c_{u,\varrho} \in \mathbf{C}_u$ indica la offerta di risorsa ϱ per l'istanza dell'operatore.

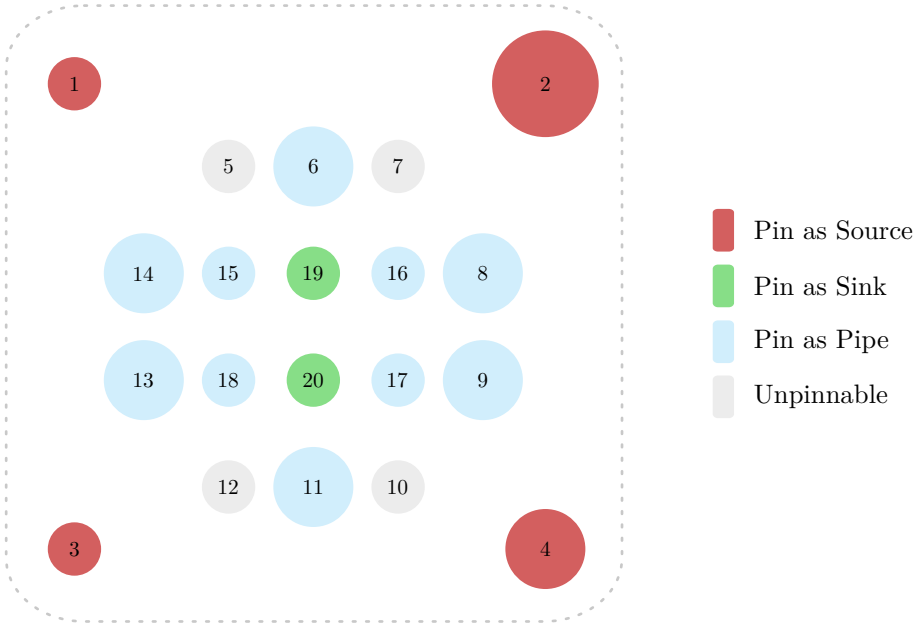


Figura 2.2: Esempio di grafo RES. Ogni nodo è etichettato con il proprio identificativo. La dimensione di un nodo u sta ad indicare la relativa offerta computazionale, in termini di risorse offerte $\mathbf{C}_u$ e speedup S_u. Il riquadro sta ad indicare la completa connettività del grafo. La distanza tra i nodi u, v sta ad indicare la relativa distanza di rete, in termini di round-trip-time $D_{(u,v)}$ e capacità del canale trasmissivo $B_{(u,v)}$. I nodi «Pin As Source» sono quei nodi $u \in V_{res} : \exists i \in So.u \in V^i_{res}$. I nodi «Pin As Sink» sono quei nodi $u \in V_{res} : \exists i \in Si.u \in V^i_{res}$. I nodi «Pin As Operator» sono quei nodi $u \in V_{res} : \exists i \in V_{dsp} \setminus So \cap Si.u \in V^i_{res}$. I nodi «Unpinnable» sono quei nodi $u \in V_{res} : \nexists i \in V_{dsp}.u \in V^i_{res}$.

S_u — Velocità di processamento delle tuple *rispetto ad un processore di riferimento*[1]. Trattandosi di un rapporto fra velocità di processamento, risulta essere adimensionale.

A_u — Probabilità di stato up-and-running.

2.2.2 Link logici

Un generico arco $(u, v) \in E_{res}$ è detto «link logico», in quanto è caratterizzato da:

[1] Una velocità $S_u = 2$ permetterà di processare il doppio delle tuple, una velocità $S_u = 0,5$ permetterà di processare la metà delle tuple, e così via.

$D_{(u,v)}$ Round Trip Time misurato sul canale. Misurato in secondi ($[s]$).

$B_{(u,v)}$ Capacità di trasferimento del canale. Misurata in bit al secondo ($[b/s]$).

$A_{(u,v)}$ Probabilità di stato up-and-running.

Connettività di rete Un link logico esprime la connettività di rete tra risorse computazionali, siano esse remote o locali. Esso può infatti essere la rappresentazione di un collegamento fisico tra risorse computazionali remote (in tal caso parleremo di *connettività remota*), o essere la rappresentazione di una tecnica IPC applicata a risorse locali (in tal caso parleremo di *connettività locale*). La connettività remota, rappresentata da archi (u, v) con $u \neq v$, rappresenta dunque la connettività di rete per nodi operazionali posizionati su nodi computazionali distinti. La connettività locale, rappresentata da archi (u, u), rappresenta la connettività di rete per nodi operazionali posizionati su uno stesso nodo computazionale.

Il grafo G_{res} presenta dunque archi autoentranti (u, u) per ogni nodo $u \in V_{res}$. Assumiamo per semplicità, che un link logico esprimente una connettività locale non apporti alcun ritardo alla trasmissione del data stream ($RTT_{(u,u)} = 0$), che possa attuare un trasferimento istantaneo ($B_{(u,u)} = \infty$) e che sia sempre disponibile ($A_{(u,u)} = 1$).

2.3 Posizionamento

Il problema del posizionamento degli operatori consiste nel mappare la topologia operazionale, rappresentata dal grafo G_{dsp}, nella topologia computazionale, rappresentata dal grafo G_{res}. Una mappatura di un grafo su un altro vuol dire associare ogni nodo dell'uno su un nodo dell'altro. Ci chiediamo dunque quale sia il modo migliore[2] di posizionare ogni nodo operazionale $i \in V_{dsp}$ su un nodo computazionale $u \in V_{res}$. Il problema del posizionamento degli operatori è un problema di ottimizzazione combinatoria riducibile al *Binary Knapsack Problem (BKP) [35, 36]* ed al *Bin Packing Problem (BPP)*

[2]la qualità di una soluzione ad un problema di ottimizzazione è quantificata da un criterio di ottimizzazione basato su uno spazio metrico. Descriveremo nel seguito lo *spazio metrico* (Sezione 2.4) ed il *criterio di ottimizzazione* (Sezione 2.5) adottati nel nostro modello.

[39]: esso appartiene pertanto alla classe di complessità computazionale NP-hard. Risulta dunque corretto, oltre che intuitivo, esprimerlo mediante un modello di *programmazione lineare binaria (PL01)* [38]. Un posizionamento degli operatori è dunque una funzione $\chi : V_{dsp} \times V_{res} \to \{0, 1\}$. Esprimiamo tale funzione mediante notazione matriciale.

Definizione 2.3. Un *posizionamento degli operatori* è una matrice binaria $|V_{dsp}| \times |V_{res}|$

$$\mathbf{x} \coloneqq (x_{i,u})_{\, i \in V_{dsp}, u \in V_{res}} \in \{0, 1\}^{|V_{dsp}||V_{res}|} \tag{2.4}$$

la cui generica componente $x_{i,u}$ esprime il posizionamento del nodo operazionale $i \in V_{dsp}$ sul nodo computazionale $u \in V_{res}$. Qualora il posizionamento sussista, abbiamo $x_{i,u} = 1$ e diciamo che l'operatore op_i «è processato» dal nodo computazionale u, altrimenti abbiamo $x_{i,u} = 0$.

Esempio 2.1. Consideriamo cinque nodi operazionali e sei nodi computazionali, tralasciando per il momento qualsiasi dettaglio topologico.

$$\mathbf{x_1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad \mathbf{x_2} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Nel posizionamento $\mathbf{x_1}$ ogni operatore è processato da un nodo computazionale distinto; in particolare, ogni operatore i è consumato da un nodo computazionale $u = i$. Nel posizionamento $\mathbf{x_2}$ gli operatori $1, 5$ sono processati rispettivamente dai nodi computazionali $1, 5$, mentre gli operatori $2, 3, 4$ sono processati dal nodo computazionale 3.

Il posizionamento degli operatori induce un naturale posizionamento dei data stream. Se infatti tra due nodi operazionali i e j, posizionati rispettivamente su nodi computazionali u e v, fluisce un data stream (i, j), questo dovrà necessariamente essere trasmesso sul link logico (u, v). E' dunque conveniente introdurre una rappresentazione vettoriale del posizionamento dei data stream. Vedremo nel seguito come tale rappresentazione, apparentemente ridondante, possa risultare comoda nella formulazione delle metriche considerate nel nostro modello (Sezione 2.4).

Definizione 2.4. Un *posizionamento dei data stream* è una matrice binaria $|E_{dsp}| \times |E_{res}|$

$$\mathbf{y} \coloneqq \left(y_{(i,j)(u,v)}\right)_{(i,j)\in E_{dsp}.(u,v)\in E_{res}} \in \{0,1\}^{|E_{dsp}||E_{res}|} \tag{2.5}$$

la cui generica componente $y_{(i,j)(u,v)}$ esprime il posizionamento del data stream $(i,j) \in E_{dsp}$ sul link logico $(u,v) \in E_{res}$. Qualora il posizionamento sussista, abbiamo $y_{(i,j)(u,v)} = 1$ e diciamo che il data stream (i,j) «fluisce» o «è trasmesso» sul link logico (u,v), altrimenti abbiamo $y_{(i,j)(u,v)} = 0$.

2.3.1 Ammissibilità

Indichiamo ora i vincoli che permettono di definire l'insieme X delle soluzioni ammissibili per il problema del posizionamento degli operatori. Poichè ogni vincolo costituisce una condizione necessaria non sufficiente alla ammissibilità della soluzione, una soluzione $\mathbf{x}$ è ammissibile, ovvero $\mathbf{x} \in X$, se e solo se soddisfa contemporaneamente ogni vincolo di ammissibilità.

Vincolo di eligibilità degli operatori Ad ogni nodo operazionale $i \in V_{dsp}$ è associato un insieme $V_{res}^i \subseteq V_{res}$ di nodi computazionali sui quali può essere posizionato. Possiamo dunque escludere a priori alcuni posizionamenti degli operatori $x_{i,u}$. Il seguente *vincolo di eligibilità* è pertanto una condizione necessaria non sufficiente alla ammissibilità del posizionamento $\mathbf{x}$:

$$u \notin V_{res}^i \Rightarrow x_{i,u} = 0 \qquad \forall i \in V_{dsp}, u \in V_{res} \tag{2.6}$$

Volendo esprimere tale vincolo in forma algebrica, otteniamo:

$$\sum_{u\in V_{res}/V_{res}^i} x_{i,u} = 0 \qquad \forall i \in V_{dsp} \tag{2.7}$$

Quanto appena asserito equivale a restringere la dimensione del posizionamento degli operatori. Diamo dunque una definizione di posizionamento degli operatori, alternativa alla 2.4:

$$\mathbf{x} \coloneqq (x_{i,u})_{i\in V_{dsp},u\in V_{res}^i} \in \{0,1\}^{|V_{dsp}|\max\{|V_{res}^i|\}} \tag{2.8}$$

Risulta dunque equivalente considerare il posizionamento nella forma standard indicata in (2.4) vincolata da (2.7), oppure considerare il posizionamento nella forma ristretta indicata in (2.8).

Vincolo di eligibilità dei data stream Dal vincolo di eligibilità degli operatori (2.6) segue che ad ogni data stream $(i,j) \in E_{dsp}$ sia associato un insieme $V^i_{res} \times V^j_{res} \subseteq E_{res}$ di link logici sui quali può essere posizionato. Possiamo dunque escludere a priori alcuni posizionamenti dei data stream $y_{(i,j)(u,v)}$. Il seguente *vincolo di eligibilità* è pertanto una condizione necessaria non sufficiente alla ammissibilità del posizionamento $\mathbf{y}$:

$$u \notin V^i_{res} \vee v \notin V^j_{res} \Rightarrow y_{(i,j)(u,v)} = 0 \qquad \forall (i,j) \in E_{dsp}, (u,v) \in E_{res} \tag{2.9}$$

Notiamo che quanto appena asserito equivale a restringere la dimensione del posizionamento dei data stream. Diamo dunque una definizione alternativa alla (2.5):

$$\mathbf{y} \coloneqq \left(y_{(i,j)(u,v)}\right)_{(i,j)\in E_{dsp}.(u,v)\in V^i_{res}\times V^j_{res}} \in \{0,1\}^{|E_{dsp}|\max\{|V^i_{res}\times V^j_{res}|\}} \tag{2.10}$$

Risulta dunque equivalente considerare il posizionamento nella forma standard indicata in (2.5) vincolata da (2.9), oppure considerare il posizionamento nella forma ristretta indicata in (2.8).

Vincolo di connettività Dalle definizioni di posizionamento risulta evidente uno stretto legame tra il posizionamento degli operatori ed il posizionamento dei data stream. Qualora i nodi operazionali i e j siano stati posizionati sui nodi computazionali u e v, il data stream (i,j) deve necessariamente essere posizionato sul link logico (u,v). Allo stesso modo, qualora il data stream (i,j) sia stato posizionato sul link logico (u,v), i nodi operazionali i e j devono necessariamente essere posizionati sui nodi computazionali u e v. Dato un posizionamento degli operatori $\mathbf{x}$ ed un posizionamento dei data stream $\mathbf{y}$, questi devono soddisfare per costruzione il seguente *vincolo di connettività*, il quale è pertanto una condizione necessaria non sufficiente alla ammissibilità del posizionamento $\mathbf{x}$:

$$y_{(i,j)(u,v)} \iff x_{i,u} \wedge x_{j,v} \qquad \forall i,j \in V_{dsp}, \forall u,v \in V_{res} \tag{2.11}$$

Volendo esprimere tale vincolo in forma algebrica, otteniamo:

$$\begin{cases} x_{i,u} + x_{j,v} - 1 \leq y_{(i,j)(u,v)} & \forall (i,j) \in E_{res}, (u,v) \in E_{dsp} \\ x_{i,u} + x_{j,v} \geq 2y_{(i,j)(u,v)} & \forall (i,j) \in E_{res}, (u,v) \in E_{dsp} \end{cases} \tag{2.12}$$

In alternativa alla 2.12, il vincolo di connettività 2.11 può essere espresso algebricamente nella forma suggerita dal vincolo di conservazione del flusso presente nella formulazione del *Metric Labeling Problem (MLP)* [39]. Tale formulazione alternativa è preferibile alla precedente, in quanto sperimentalmente più efficiente.

$$\begin{cases} x_{i,u} = \sum_{(i,j) \in E_{dsp}} \sum_{v \in V_{res}^j} y_{(i,j)(u,v)} & \forall i \in V_{dsp} \setminus So, u \in V_{res}^i \\ x_{j,v} = \sum_{(i,j) \in E_{dsp}} \sum_{u \in V_{res}^i} y_{(i,j)(u,v)} & \forall j \in V_{dsp} \setminus Si, v \in V_{res}^j \end{cases} \tag{2.13}$$

Tale ultima rappresentazione può tuttavia risultare contro-intuitiva: in finale, vi si preferisce pertanto la seguente forma:

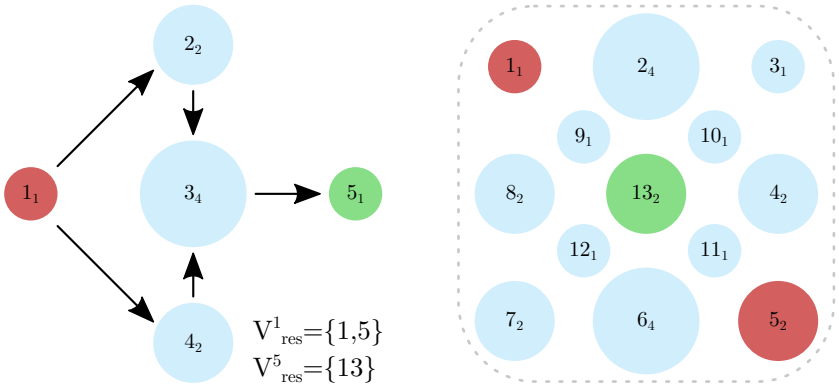
$$\begin{cases} x_{i,u} = \sum_{v \in V_{res}^j} y_{(i,j)(u,v)} & \forall (i,j) \in E_{dsp}, u \in V_{res}^i \\ x_{j,v} = \sum_{u \in V_{res}^i} y_{(i,j)(u,v)} & \forall (i,j) \in E_{dsp}, v \in V_{res}^j \end{cases} \tag{2.14}$$

Notiamo che da quanto asserito in (2.14), segue che una soluzione al problema di posizionamento degli operatori può essere espressa sia in termini di $\mathbf{x}$ che in termini di $\mathbf{y}$, in quanto dall'uno può essere derivato l'altro. Tuttavia, nel seguito faremo riferimento ad $\mathbf{x}$ come espressione della soluzione al nostro problema di posizionamento.

Vincolo di unicità Ogni nodo operazionale deve essere posizionato su uno ed un solo nodo computazionale. Il seguente *vincolo di unicità* è pertanto una condizione necessaria non sufficiente alla ammissibilità del posizionamento $\mathbf{x}$.

$$\sum_{i \in V_{dsp}} x_{i,u} = 1 \qquad \forall u \in V_{res} \tag{2.15}$$

Vincolo di capacità L'offerta di risorse di ogni nodo computazionale deve poter soddisfare la domanda di ogni nodo operazionale in esso posizionato.



(a) (b)

Figura 2.3: Grafi di riferimento per l'Esempio 2.2. Nel grafo DSP (a) abbiamo etichettato ogni nodo con il proprio identificativo, seguito dalla relativa domanda computazionale. Se non diversamente specificato, assumiamo che $\forall i \in V_{dsp} : V^i_{res} = V_{res}$. Nel grafo RES (b) abbiamo etichettato ogni nodo con il proprio identificativo, seguito dalla relativa offerta computazionale. Il riquadro sta ad indicare la completa connettività del grafo.

Il seguente *vincolo di capacità* è pertanto una condizione necessaria non sufficiente alla ammissibilità del posizionamento $\mathbf{x}$.

$$\sum_{i \in V_{dsp}} x_{i,u} \mathbf{C}_i \leq \mathbf{C}_u \qquad \forall u \in V_{res} \tag{2.16}$$

Sebbene possa risultare evidente, precisiamo che in (2.16) abbiamo assunto che $\mathbf{C}_i \leq \mathbf{C}_u \Leftrightarrow \forall c_{i,\varrho} \in \mathbf{C}_i, c_{u,\varrho} \in \mathbf{C}_u . c_{i,\varrho} \leq c_{u,\varrho}$.

Esempio 2.2. Consideriamo la applicazione DSP e la architettura distribuita indicate in Figura 2.3. Studiamo la ammissibilità dei posizionamenti $\mathbf{x_1}$ ed $\mathbf{x_2}$.

$$\mathbf{x_1} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{bmatrix}$$

$$\mathbf{x_2} = \begin{bmatrix} 1 & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 0 \end{bmatrix}$$

Il posizionamento $\mathbf{x_1}$ è ammissibile, mentre il posizionamento $\mathbf{x_2}$ non lo è. Il posizionamento $\mathbf{x_2}$ viola infatti il vincolo di unicità (riga 1 e 2), il vincolo di capacità (riga 3) ed il vincolo di eligibilità degli operatori (riga 5).

2.4 Metriche

Dato un insieme di soluzioni ammissibili X, la qualità di una soluzione $x \in X$ è quantificata da un vettore $\mathbf{M}(x)$ in uno spazio metrico $\mathcal{M}$ le cui dimensioni sono le metriche m_i considerate nel problema. Indichiamo con M l'insieme di tali metriche. Ogni metrica $m \in M$ assume la forma di una funzione $f_m : X \to A$, dove A è un generico codominio. L'obiettivo di una metrica è il criterio con cui questa debba essere ottimizzata: questo può dunque essere una massimizzazione $\max_{x \in X} f_m(x)$ o una minimizzazione $\min_{x \in X} f_m(x)$.

Definizione 2.5. Due metriche m_1 ed m_2 sono *antitetiche* se caratterizzate da obiettivi opposti, ovvero se l'una deve essere massimizzata e l'altra minimizzata, e viceversa. Un insieme di metriche M è antitetico se contiene almeno due metriche antitetiche. Uno spazio metrico $\mathcal{M}$ è antitetico se le sue dimensioni sono elementi di un insieme antitetico. In caso contrario parliamo di metriche omotetiche, insieme di metriche omotetico e spazio metrico omotetico.

Se la qualità di una soluzione è valutata da metriche antitetiche, questa deve necessariamente considerare un *trade-off* che esprima la priorità di una metrica rispetto all'altra.

Definizione 2.6. Due metriche m_1 ed m_2 sono *disomogenee* se per esse non ha senso definire un ordinamento totale. Un insieme di metriche M è disomogeneo se contiene almeno due metriche disomogenee. Uno spazio

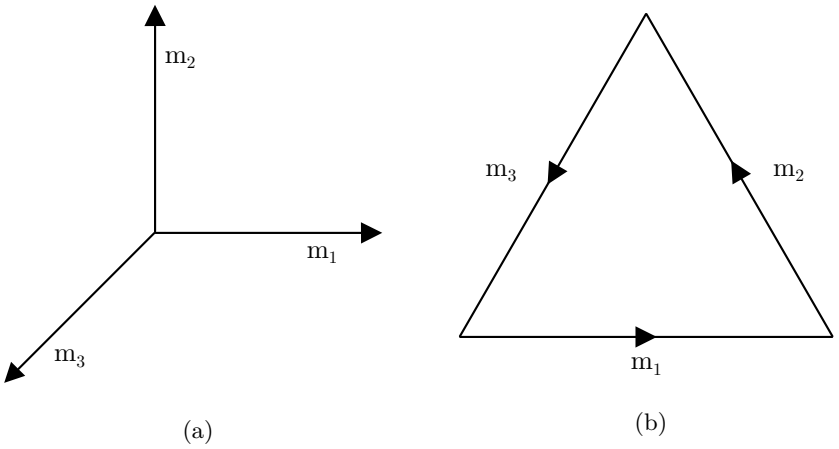


Figura 2.4: Spazio metrico omotetico (a) e spazio metrico antitetico (b).

metrico $\mathcal{M}$ è disomogeneo se le sue dimensioni sono elementi di un insieme disomogeneo. In caso contrario, parliamo di metriche omogenee, insieme di metriche omogeneo e spazio metrico omogeneo.

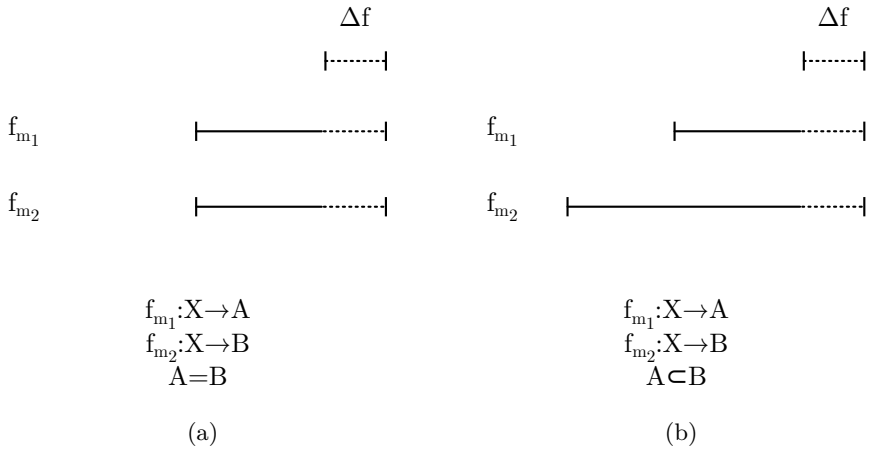


Figura 2.5: Metriche omogenee (a) e metriche disomogenee (b).

Due metriche sono dunque disomogenee se rappresentate da funzioni aventi codominii differenti. Qualora infatti una metrica m_1 assuma la forma di una funzione $f_{m_1} : X \to A$ ed una metrica m_2 assuma la forma di una funzione $f_{m_2} : X \to B \subset A$, variazioni eguali $\triangle f_{m_1}$ e $\triangle f_{m_2}$ non sono tra loro confrontabili. Si pensi, ad esempio, ad una metrica m_1 che assuma valore

in tutto $\mathbb{R}$ e ad una metrica m_2 che assuma valore in $[0, 1]$: una variazione $\triangle f_{m_2} = 0.5$ non ha lo stesso significato di una variazione $\triangle f_{m_1} = 0.5$.

Se la qualità di una soluzione è valutata da metriche disomogenee, questa deve necessariamente considerare una *normalizzazione* delle metriche che le renda confrontabili.

2.4.1 Metriche del modello

Nel nostro modello consideriamo uno spazio metrico antitetico e disomogeneo (Definizione 2.5 e Definizione 2.6), le cui dimensioni sono il tempo di risposta della applicazione DSP e la disponibilità della architettura distribuita. Il tempo di risposta della applicazione è di fondamentale importanza, in quanto è il fattore che contribuisce immediatamente alla qualità percepita dall'utente finale. Esso assume un valore particolarmente importante quando si consideri una architettura Grid. Una tale architettura introduce infatti necessariamente delle latenze di comunicazione crescenti con il suo grado di distribuzione. La disponibilità delle risorse computazionali e trasmissive della architettura è di fondamentale importanza per la minimizzazione delle procedure di load shedding, le quali introducono latenze di processamento sia in una architettura Cloud che in una architettura Grid.

2.4.1.1 Tempo di risposta

Il tempo di risposta R di una applicazione DSP è il tempo worst case necessario affinchè il data stream prodotto da un nodo source possa essere consumato da un nodo sink. Esaminiamo nel dettaglio le componenti di questa metrica.

Dato un posizionamento degli operatori $\mathbf{x}$, il generico posizionamento $x_{i,u}$ induce un *ritardo di esecuzione* $R_{ex}(i, u)$, dovuto al processamento dell'operatore op_i sul nodo computazionale u.

$$R_{ex}(i, u) \coloneqq \frac{\varphi_i^{in}}{S_u S_i} \qquad \forall i \in V_{dsp}, u \in V_{res} \tag{2.17}$$

Dato un posizionamento dei data stream $\mathbf{y}$, il generico posizionamento $y_{(i,j)(u.v)}$ induce un *ritardo di trasferimento* $R_{tx}((i, j), (u, v))$, dovuto alla trasmissione

del data stream (i, j) sul link logico (u, v).

$$R_{tx}((i,j),(u,v)) \coloneqq \frac{\varphi_{(i,j)}\lambda_i^{out}}{B_{(u,v)}} + D_{(u,v)} \qquad \forall (i,j) \in E_{dsp}, (u,v) \in E_{res} \tag{2.18}$$

Dato un cammino $\pi \in \Pi_{G_{dsp}}$ ed un posizionamento degli operatori $\mathbf{x}$, il *tempo di risposta associato al cammino* $R_\pi(\mathbf{x})$ è composto dai ritardi di esecuzione dei nodi visitati e dai ritardi di trasmissione degli archi attraversati.

$$\begin{aligned} R_\pi(\mathbf{x}) \coloneqq & \sum_{i \in V_{dsp}} \sum_{u \in V_{res}^i} x_{i,u} R_{ex}(i,u) + \\ & \sum_{k=1}^{n_\pi - 1} \sum_{(u,v) \in V_{res}^{i_k} \times V^{i_k+1}} y_{(i_k,i_{k+1})(u,v)} R_{tx}((i,j),(u,v)) \\ & \forall \pi = (i_1, \ldots, i_{n_\pi}) \in \Pi_{dsp} \end{aligned} \tag{2.19}$$

Definizione 2.7. Dato un posizionamento degli operatori $\mathbf{x}$, il tempo di risposta $R(\mathbf{x})$ di una applicazione DSP è il tempo worst case necessario affinchè il data stream prodotto da un nodo source possa essere consumato da un nodo sink.

$$R(\mathbf{x}) \coloneqq \max_{\pi \in \Pi_{dsp}} R_\pi(\mathbf{x}) \tag{2.20}$$

2.4.1.2 Disponibilità

La disponibilità A di una architettura distribuita è la probabilità di eseguire il processo di trasformazione del data stream assegnandone l'esecuzione a nodi computazionali up-and-running. Esaminiamo nel dettaglio le componenti di questa metrica.

Dato un posizionamento dei data stream $\mathbf{y}$, il generico posizionamento $y_{(i,j)(u.v)}$ induce una *disponibilità di esecuzione* dovuta alla esecuzione degli operatori nei nodi computazionali in cui sono posizionati.

$$A_{ex}(\mathbf{x}) \coloneqq \prod_{i \in V_{dsp}} \left(\sum_{u \in V_{res}^i} x_{i,u} A_u \right) \tag{2.21}$$

Dato un posizionamento dei data stream $\mathbf{y}$, il generico posizionamento $y_{(i,j)(u.v)}$ induce una *disponibilità di trasferimento* dovuta alla trasmissione dei data stream sui link logici.

$$A_{tx}(\mathbf{y}) \coloneqq \prod_{(i,j)\in E_{res}} \left(\sum_{(u,v)\in V^i_{res}\times V^j_{res}} y_{(i,j)(u,v)} A_{(u,v)} \right) \tag{2.22}$$

Definizione 2.8. Dato un posizionamento degli operatori $\mathbf{x}$, la disponibilità $A(\mathbf{x})$ di una architettura distribuita è la probabilità di eseguire il processo di trasformazione del data stream assegnandone l'esecuzione a nodi computazionali up-and-running.

$$A(\mathbf{x}) \coloneqq A_{ex}(\mathbf{x}) \cdot A_{tx}(\mathbf{y}) \tag{2.23}$$

In quanto appena definito abbiamo assunto per semplicità, l'indipendenza delle disponibilità di nodi computazionali e link logici. E' chiaro che, da un punto di vista di evoluzione del traffico di rete, la disponibilità di un link logico debba essere correlata alla disponibilità delle risorse di rete connesse da tale link logico. Tuttavia, il livello di astrazione a cui fa riferimento il nostro modello rende lecito e conveniente assumere l'indipendenza stocastica di questi eventi.

Onde evitare il dispendio computazionale dovuto alle produttorie in (2.21) e (2.22), diamo una formulazione alternativa della (2.23), risultante dalla applicazione delle proprietà dei logaritmi[3].

$$\log A(\mathbf{x}) = \sum_{i\in V_{dsp}} \left(\sum_{u\in V^i_{res}} x_{i,u} \log A_u \right) + \sum_{(i,j)\in E_{res}} \left(\sum_{(u,v)\in V^i_{res}\times V^j_{res}} y_{(i,j)(u,v)} \log A_{(u,v)} \right) \tag{2.24}$$

[3]Precisiamo che la proprietà dei logaritmi è soddisfatta in quanto, per il vincolo di unicità (2.15), abbiamo che $\log \sum_{u\in V^i_{res}} x_{i,u} A_u = \sum_{u\in V^i_{res}} x_{i,u} \log A_u$ e $\log \sum_{(u,v)\in V^i_{res}\times V^j_{res}} y_{(i,j)(u,v)} A_{(u,v)} = \sum_{(u,v)\in V^i_{res}\times V^j_{res}} y_{(i,j)(u,v)} \log A_{(u,v)}$

2.5 Ottimizzazione

La selezione di una soluzione ammissibile $x \in X$ da parte di un decisore deve essere orientata da un *criterio di ottimizzazione*, sia essa una minimizzazione o una massimizzazione di una opportuna funzione obiettivo $F(x)$. Qualora tale criterio aggreghi più di una metrica $m \in M$, si parla di *ottimizzazione multi-obiettivo*, o più in generale di *problema di ottimizzazione multi-obiettivo*[4].

Il problema del posizionamento degli operatori da noi formulato è un problema multi-obiettivo con insieme di metriche antitetiche (Definizione 2.5) e disomogenee (Definizione 2.6). Come suggerito in [38], ci serviremo dell'*approccio di Čebyšëv* per ricavare la formulazione del criterio di ottimizzazione del nostro modello.

2.5.1 Approccio di Čebyšëv

Introduciamo la nozione della norma di Čebyšëv, mostrando come questa suggerisca un utile approccio alla formulazione di un criterio di ottimizzazione in problemi di ottimizzazione multi-obiettivo [42], che prende pertanto il nome di *approccio di Čebyšëv ai problemi di ottimizzazione*[5].

Definizione 2.9. Dato uno spazio vettoriale N-dimensionale, la *norma di Čebyšëv* tra due punti p e q, aventi coordinate standard $p_{n\in N}$ e $q_{n\in N}$, è definita come:

$$dist_{Cebishev}(p, q) \coloneqq \max_{n\in N} \{|p_n - q_n|\} \tag{2.25}$$

La nozione di norma di Čebyšëv suggerisce di minimizzare la massima distanza tra valori reali, ovvero quelli ottenuti da una soluzione, e valori utopici, ovvero quelli ideali desiderati.

Definizione 2.10. Siano dati un insieme di soluzioni ammissibili X ed un insieme di metriche M. Ogni metrica $m \in M$ assuma la forma di una

[4] i *problemi di ottimizzazione multi-obiettivo (Multi-objective Optimization Problems, MOPs)* si incontrano spesso nello studio di sistemi complessi. La *dominanza di Pareto* è il criterio tipicamente utilizzato per valutare sia soluzioni a problemi di ottimizzazione singolo-obiettivo che soluzioni MOPs. Tuttavia, qualora un MOP consideri più di tre obiettivi (parliamo in tal caso di *Many-objective Optimization Problems, MaOPs*), la dominanza di Pareto risulta non essere più un criterio applicabile [42].

[5] anche detto approccio MOP (Multi-objective Optimization Problem) di Čebyšëv.

una funzione $f_m : X \to \mathbb{R}$. Per ogni metrica sia definito un valore utopico $\hat{f}_m$. Ad ogni metrica sia inoltre associato un peso $w_m \in \mathbb{R}^+$ tale che $\sum_{m \in M} w_m = 1$, il quale esprime il trade-off tra una ottimizzazione guidata dall'una o dall'altra metrica. L'approccio di Čebyšëv ai problemi di ottimizzazione multi-obiettivo consiste nella seguente formulazione del criterio di ottimizzazione:

$$\min_{x \in X} \max \left\{ w_m | f_m(x) - \hat{f}_m | \right\}_{m \in M} \tag{2.26}$$

2.5.2 Funzione obiettivo del modello

Il problema del posizionamento degli operatori da noi formulato è un problema multi-obiettivo con insieme di metriche antitetiche (Definizione 2.5) e disomogenee (Definizione 2.6). L'approccio di Čebyšëv (2.26) può certamente essere applicato ad uno spazio metrico antitetico, ma non ad uno spazio metrico disomogeneo. Inoltre, esso presenta due passi di ottimizzazione: una massimizzazione innestata in una minimizzazione. Esso può dunque essere un ottimo punto di partenza per la formulazione del nostro criterio di ottimizzazione, ma devono essere apportate delle modifiche.

Esprimiamo la (2.26) di modo che possa essere formulata con un unico passo di ottimizzazione, ovvero con una massimizzazione semplice. A tal fine partizioniamo l'insieme delle metriche M nell'insieme M_1 delle metriche da minimizzare ed M_2 delle metriche da massimizzare. Otteniamo:

$$\max_{x \in X} \sum_{m \in M_1} w_m \left(f_m^{max} - f_m(x) \right) + \sum_{m \in M_2} w_m \left(f_m(x) - f_m^{min} \right) \tag{2.27}$$

dove f_m^{max} è il valore ideale di una metrica da minimizzare, ed f_m^{min} è il valore ideale di una metrica da massimizzare.

Qualora M sia un insieme di metriche antitetiche e disomogenee, le funzioni f_m devono essere normalizzate in modo da permettere una definizione significativa di un ordinamento totale. Otteniamo:

$$\max_{x \in X} \sum_{m \in M_1} w_m \frac{f_m^{max} - f_m(x)}{f_m^{max} - f_m^{min}} + \sum_{m \in M_2} w_m \frac{f_m(x) - f_m^{min}}{f_m^{max} - f_m^{min}} \tag{2.28}$$

dove f_m^{max} ed f_m^{min} sono rispettivamente l'estremo superiore ed inferiore della normalizzazione. Notiamo che, in seguito alla normalizzazione, ogni metrica è compesa in una intervallo $[0, 1]$, dove 0 corrisponde al worst case ed 1 corrisponde al best case.

In generale dunque, la funzione obiettivo di un problema multi-obiettivo con spazio metrico M antitetico e disomogeneo assume la seguente forma da massimizzare:

$$F(x) = \sum_{m \in M_1} w_m \frac{f_m^{max} - f_m(x)}{f_m^{max} - f_m^{min}} + \sum_{m \in M_2} w_m \frac{f_m(x) - f_m^{min}}{f_m^{max} - f_m^{min}} \tag{2.29}$$

Ci serviamo della (2.29) per formulare la funzione obiettivo del nostro modello:

$$F(\mathbf{x}) = w_R \frac{R_{max} - R(\mathbf{x})}{R_{max} - R_{min}} + w_A \frac{\log A(\mathbf{x}) - \log A_{min}}{\log A_{max} - \log A_{min}} \tag{2.30}$$

I pesi $w_R, w_A \in \mathbb{R}^+$ tali che $w_R + w_A = 1$, esprimono il trade-off tra una ottimizzazione guidata dalla minimizzazione del tempo di risposta ed una guidata dalla massimizzazione della disponibilità. I valori R_{max} ed R_{min} denotano rispettivamente il massimo ed il minimo tempo di risposta atteso. I valori A_{max} ed A_{min} denotano rispettivamente la massima e la minima disponibilità attesa.

Funzione obiettivo semplificata La funzione obiettivo espressa in (2.30) ha lo svantaggio di pretendere la conoscenza di quattro valori: R_{max}, R_{min}, A_{max} e A_{min}. Formuliamo dunque una funzione obiettivo semplificata, nella quale sia necessaria la conoscenza di solo due valori. Tale funzione semplificata assume la seguente forma da massimizzare:

$$\tilde{F}(\mathbf{x}) = -w_R \frac{R(\mathbf{x})}{\tilde{R}_{max}} - w_A \frac{\log A(\mathbf{x})}{\log \tilde{A}_{min}} \tag{2.31}$$

dove $\tilde{R}_{max}$ è il massimo tempo di risposta accettato e $\tilde{A}_{min}$ è la minima disponibilità accettata.

2.6 Formulazioni

Concludiamo questo capitolo dando le formulazioni in programmazione lineare binaria del nostro modello per il problema del posizionamento degli operatori.

2.6.1 Modello OPPStandard

$$
\begin{array}{lll}
\textbf{maximize}: & F(\mathbf{x}) & \\
\textbf{subject to}: & & \\
\sum_{i \in V_{dsp}} x_{i,u}\mathbf{C}_i & \leq & \mathbf{C}_u \qquad \forall u \in V_{res} \\
\sum_{u \in V_{res}} x_{i,u} & = & 1 \qquad \forall i \in V_{dsp} \\
\sum_{u \in V_{res} \setminus V^i_{res}} x_{i,u} & = & 0 \qquad \forall i \in V_{dsp} \\
x_{i,u} + x_{j,v} - 1 & \leq & y_{(i,j)(u,v)} \qquad \forall (i,j) \in E_{res}, (u,v) \in E_{dsp} \\
x_{i,u} + x_{j,v} & \geq & 2y_{(i,j)(u,v)} \qquad \forall (i,j) \in E_{res}, (u,v) \in E_{dsp} \\
x_{i,u} & \in & \{0,1\} \qquad \forall i \in V_{dsp}, u \in V_{res} \\
y_{(i,j)(u,v)} & \in & \{0,1\} \qquad \forall (i,j) \in E_{dsp}, (u,v) \in E_{res}
\end{array}
\tag{2.32}
$$

2.6.2 Modello OPPRestricted

Forniamo inoltre una formulazione alternativa alla (2.32), la quale considera il posizionamento ristretto degli operatori (2.8) e dei data stream (2.10).

$$
\begin{array}{lll}
\textbf{maximize}: & F(\mathbf{x}) & \\
\textbf{subject to}: & & \\
\sum_{i \in V_{dsp}} x_{i,u}\mathbf{C}_i & \leq & \mathbf{C}_u \qquad \forall u \in V_{res} \\
\sum_{u \in V^i_{res}} x_{i,u} & = & 1 \qquad \forall i \in V_{dsp} \\
x_{i,u} + x_{j,v} - 1 & \leq & y_{(i,j)(u,v)} \qquad \forall (i,j) \in E_{res}, (u,v) \in E_{dsp} \\
x_{i,u} + x_{j,v} & \geq & 2y_{(i,j)(u,v)} \qquad \forall (i,j) \in E_{res}, (u,v) \in E_{dsp} \\
x_{i,u} & \in & \{0,1\} \qquad \forall i \in V_{dsp}, u \in V^i_{res} \\
y_{(i,j)(u,v)} & \in & \{0,1\} \qquad \forall (i,j) \in E_{dsp}, (u,v) \in V^i_{res} \times V^j_{res}
\end{array}
\tag{2.33}
$$

2.6.3 Modello OPPConservative

Forniamo inoltre una formulazione alternativa alla (2.32), la quale considera il posizionamento ristretto degli operatori (2.8) e dei data stream (2.10), nonchè il vincolo di connettività espresso secondo la conservazione del flusso del Metric Labeling Problem (2.13).

$$
\begin{array}{lll}
\textbf{maximize}: & F(\mathbf{x}) & \\
\textbf{subject to}: & & \\
\displaystyle\sum_{i \in V_{dsp}} x_{i,u}\mathbf{C}_i & \leq & \mathbf{C}_u \qquad \forall u \in V_{res} \\
\displaystyle\sum_{u \in V^i_{res}} x_{i,u} & = & 1 \qquad \forall i \in V_{dsp} \\
\displaystyle\sum_{v \in V^j_{res}} y_{(i,j)(u,v)} & = & x_{i,u} \qquad \forall (i,j) \in E_{dsp}, u \in V^i_{res} \\
\displaystyle\sum_{u \in V^i_{res}} y_{(i,j)(u,v)} & = & x_{j,v} \qquad \forall (i,j) \in E_{dsp}, v \in V^j_{res} \\
x_{i,u} & \in & \{0,1\} \qquad \forall i \in V_{dsp}, u \in V^i_{res} \\
y_{(i,j)(u,v)} & \in & \{0,1\} \quad \forall (i,j) \in E_{dsp}, (u,v) \in V^i_{res} \times V^j_{res}
\end{array}
\tag{2.34}
$$

Capitolo 3

Implementazione

In questo capitolo descriviamo brevemente l'implementazione software dei modelli proposti per il problema di posizionamento. Nel seguito, parleremo dunque di "modello" in riferimento all'implementazione software della corrispondente formulazione matematica.

I modelli sono stati realizzati dapprima in Optimization Programming Language (OPL) all'interno dell'ambiente di sviluppo integrato IBM ILOG CPLEX Optimization Studio. Abbiamo adottato questa soluzione preliminare in quanto era necessaria una rapida prototipazione dei modelli, volta a valutarne le prestazioni su piccole istanze ad hoc.

Successivamente, questi sono stati implementati in Java, dando vita al progetto OPMap. Questa soluzione ci ha permesso di condurre un'analisi sperimentale quanto più possibile estensiva sui modelli in esame. La metodologia e i risultati di questa analisi sono oggetto del prossimo capitolo.

3.1 IBM ILOG CPLEX Optimization Studio

IBM ILOG[1] CPLEX[2] Optimization Studio è un ambiente di sviluppo integrato (IDE) per la creazione, risoluzione ed analisi di modelli di ottimizzazio-

[1]ILOG è una multinazionale fracese leader nel settore dei Business Process Management Systems (BPMS), acquisita da IBM nel Gennaio 2009 per 340M$ [43]. Uno dei prodotti che motivarono l'acquisizione fu proprio ILOG CPLEX.

[2]CPLEX prende il nome dal linguaggio di programmazione in cui è stato sviluppato (C), e dal noto algoritmo del Simplesso (Simplex) [44], primo algoritmo implementato nella libreria e pietra miliare dei metodi risolutivi per problemi di programmazione lineare. Sebbene oggi la libreria implementi anche altri algoritmi risolutivi, si è deciso di mantenere

ne [45]. Commercializzato, nella sua attuale struttura, a partire dal 2010, è divenuto presto un prodotto di punta nel settore dei Business Process Management Systems (BPMSs).

Questo IDE unisce lo stile modulare dell'ambiente di sviluppo open source Eclipse, la forte espressività del linguaggio OPL e la comprovata efficienza degli algoritmi di risoluzione ILOG CPLEX Optimizer e IBM CP Optimizer. Di anno in anno questo prodotto migliora le proprie performance nella risoluzione di modelli Mixed Integer Problems (MIPs) [46]. In generale, questi problemi sono computazionalmente più complessi rispetto alle controparti continue [38], ma sono anche quelli che più diffusamente trovano applicazione nei processi di business intelligence e di controllo industriale [47].

OPL L'Optimization Programming Language (OPL) è un linguaggio di programmazione dichiarativo sviluppato da IBM appositamente per essere utilizzato all'interno dei prodotti della famiglia CPLEX. Questo linguaggio è caratterizzato da una sintassi totalmente orientatata alla descrizione matematica di modelli di ottimizzazione, pertanto affine ai principi della Mathematical Programming (MP) e della Constraint Programming (CP) [48, 49]. Oltre alla definizione dei modelli, il linguaggio permette di definire il dataset e stabilire semplici procedure di pre/post-processamento[3].

OPL permette di minimizzare il codice necessario, massimizzandone l'espressività e rendendo l'applicazione facilmente manutenibile. Di contro, questo risulta poco flessibile in termini di strutture dati, rendendo inagevole il controllo dinamico delle istanze.

Struttura di un progetto CPLEX Un progetto CPLEX prevede almeno (i) un file (.mod) in OPL per la descrizione del modello, (ii) un file (.dat) in OPL per la definizione del dataset e (iii) un file (.ops) in XML per l'impostazione dei parametri di risoluzione. L'associazione di un file .mod, un file .dat ed un file .ops costituisce una configurazione di esecuzione. Un

il nome originario. Lo sviluppo della prima versione si deve a Robert E. Bixby, e la sua prima commercializzazione alla CPLEX Optimization Inc. nel 1988, la quale fu poi acquisita dalla ILOG nel 1997.

[3]per il pre/post-processamento OPL fornisce un liguaggio simile al Javascript, che per questo è stato chiamato OPL Script.

progetto presenta, tipicamente, più configurazioni di esecuzione alternative, al fine di confrontarne i risultati, sia in termini di obiettivo raggiunto che di prestazioni del modello. Per dettagli operativi sulla realizzazione, esecuzione ed analisi di un progetto OPL all'interno dell'IDE, rimandiamo il lettore alle guide fornite dal produttore [50, 47, 51, 52].

L'implementazione dei modelli all'interno di IBM ILOG CPLEX Optimization Studio ci ha garantito un rapido processo di prototipazione, consentendoci così di avere subito una prima valutazione delle performance. Nondimeno ci ha permesso di acquisire familiarità con le potenzialità di CPLEX.

Il codice sorgente dell'implementazione dei modelli è disponibile in [53].

3.2 OPMap

OPMap (OPerator Mapping) è l'applicazione sviluppata appositamente per condurre l'analisi sperimentale sui modelli proposti per il problema di posizionamento. L'applicazione è scritta in Java8 ed utilizza le APIs esposte da IBM Java Concert Technology per interfacciarsi con il core del motore CPLEX. In Figura 3.1 mostriamo una semplice rappresentazione dell'architettura OPMap, che descriveremo nel seguito adottando un approccio bottom-up.

CPLEX Core, o *CPLEX Callable Library*, è la libreria C che realizza tutte le funzionalità del motore CPLEX.

Java Native Interface (JNI) è il framework, sviluppato da Oracle, che realizza l'interfaccia bidirezionale tra porzioni di codice Java e porzioni di codice nativo[4] . La JNI viene dunque tipicamente utilizzata per invocare porzioni di codice C/C++ o Assembly all'interno di programmi Java [54]. Nella nostra architettura, la JNI viene utilizzata internamente da Java Concert Technology per invocare i metodi della libreria CPLEX.

Java Concert Technology è il framework Java, sviluppato da IBM, che espone le APIs necessarie alla creazione, risoluzione ed analisi dei modelli di ottimizzazione. Queste APIs forniscono un totale accesso alle funzionalità

[4]per codice «nativo» si intende un codice scritto in un linguaggio di programmazione specifico di un determinato sistema operativo, pertanto intrinsecamente non portabile.

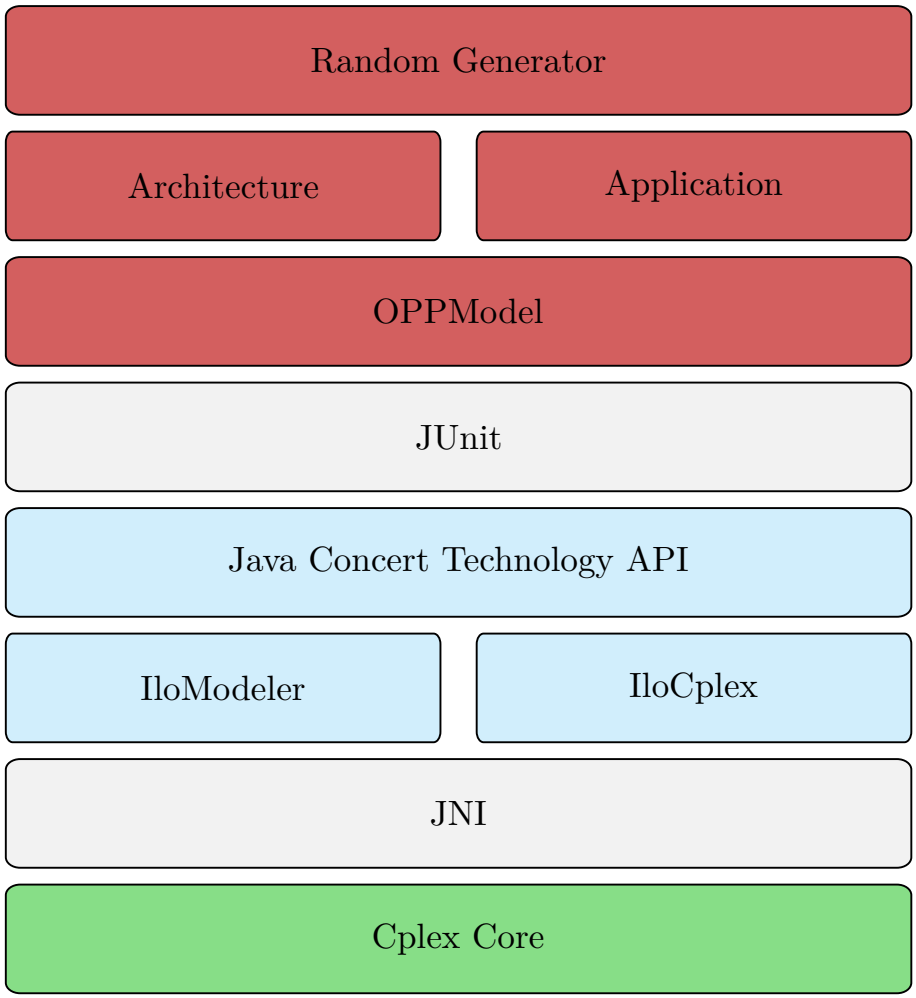


Figura 3.1: Architettura OPMap

del motore CPLEX, attraverso (i) *IloModeler*, ovvero la factory che permette di creare le componenti del modello nella forma di oggetti Java, e (ii) *IloCplex*, ovvero la classe tramite la quale è possibile risolvere ed analizzare un modello compilato. Nella nostra architettura, *IloModeler* è usato dal costruttore di ogni modello OPP per istanziarsi come modello CPLEX, mentre *IloCplex* è usato dal risolutore OPP per invocare l'algoritmo di risoluzione ed accedere al valore delle soluzioni.

JUnit [55] è un framework Java open source per la creazione di test. Nella nostra architettura, lo abbiamo utilizzato per testare la compilazione e la risoluzione dei modelli. Ogni esperimento previsto dall'analisi sperimentale è stato implementato nella forma di un *JUnit TestCase*.

OPPModel è l'interfaccia implementata da un qualsivoglia modello OPP. Il costruttore del modello riceve in input (i) una *Architecture* , ovvero l'oggetto che incapsula i dettagli dell'architettura distribuita, e (ii) una *Application*, ovvero l'oggetto che incapsula i dettagli dell'applicazione DSP. Il costruttore del modello invoca i metodi di *IloModeler* per istanziarsi come modello CPLEX.

RandomGenerator è una factory adibita alla generazione casuale di architetture distribuite ed applicazioni DSP. Questo generatore garantisce il pieno controllo statistico e deterministico delle istanze del problema, ed è pertanto impiegato in ogni esperimento per generare le istanze secondo i parametri statistici e deterministici previsti dall'esperimento.

Il codice sorgente del progetto OPMap è disponibile in [56]

Sviluppi futuri In futuro, speriamo che il progetto OPMap possa diventare un utile strumento per la prototipazione rapida in Java di modelli OPP, e che possa fornire strumenti di benchmark a supporto dello sviluppo di soluzioni euristiche al problema. Nell'ottica di questo obiettivo, la prossima release di OPMap fornirà (i) una CLI per la definizione dello scopo e dei parametri degli esperimenti, (ii) la possibilità di inserire a run-time una propria implementazione di modello OPP, (iii) una suite di esperimenti volti a valutare il consumo di memoria in fase di compilazione e risoluzione, (iv) risoluzione mediante IBM Network Optimizer, e (v) risultati degli esperimenti nella forma prevista dall'analisi boxplot.

Capitolo 4

Analisi Sperimentale

In questa sezione forniamo la metodologia e i risultati dell'analisi sperimentale condotta sui tre modelli di ottimizzazione proposti per il problema del posizionamento degli operatori. L'oggetto dell'analisi è la comparazione delle performance di ogni modello, in funzione della dimensione e distribuzione dell'input. Confrontiamo ogni modello, prendendone in esame sia il processo di compilazione che quello di risoluzione. Il tempo di risposta percepito da un agente che necessiti di una soluzione di posizionamento, non è infatti dovuto esclusivamente all'esecuzione di un algoritmo risolutivo sul modello, bensì anche dal tempo necessario alla creazione delle strutture dati adibite alla sua rappresentazione. Vedremo infatti come tale tempo di compilazione possa, sotto determinate condizioni, risultare non trascurabile.

Ricordiamo, per comodità del lettore, i fattori essenziali che distinguono un modello dall'altro.

OPPStandard il dominio delle variabili decisionali considera ogni possibile combinazione di posizionamento di nodi operazionali su nodi computazionali. La pinnabilità di un nodo operazionale su uno specifico sottoinsieme di nodi computazionali è espressa dal vincolo di eligibilità. I vincoli di connettività sono espressi come traduzione algebrica diretta del vincolo logico.

OPPRestricted il dominio delle variabili decisionali è ristretto alle sole combinazioni di posizionamento considerate ammissibili dalla pinnabilità dei nodi operazionali. Il vincolo di eligibilità viene dunque a

mancare, in quanto implicito nella definizione del dominio. I vincoli di connettività sono espressi come traduzione algebrica diretta del vincolo logico.

OPPConservative il dominio delle variabili decisionali è ristretto alle sole combinazioni di posizionamento considerate ammissibili dalla pinnabilità dei nodi operazionali. Il vincolo di eligibilità viene dunque a mancare, in quanto implicito nella definizione del dominio. I vincoli di connettività sono espressi secondo la formulazione della conservazione del flusso.

Confrontando le formulazioni matematiche dei modelli (Equazioni 2.32, 2.33, 2.34), ci si rende presto conto di come queste consistano ella modifica di quantificatori o, al più, di uno specifico vincolo. Vedremo come tali semplici differenze possano avere una forte incidenza in termini di performance, sia dal punto di vista della compilazione del modello che della sua risoluzione.

4.1 Dimensione e distribuzione dell'input

I grafi orientati dell'architettura distribuita e dell'applicazione DSP costituiscono l'input del nostro problema. Per dimesione dell'input intendiamo dunque l'ordine e la dimensione di tali grafi, ovvero la cardinalità del loro insieme di nodi e del loro insieme di archi. Per distribuzione intendiamo invece la caratterizzazione interna dei loro nodi ed archi, ovvero il valore degli attributi con i quali abbiamo stabilito di definirli.

La dimensione dell'input determina la dimensione del modello, ovvero il numero delle realizzazioni dei suoi vincoli e dei termini presenti in ognuna di esse. E' dunque prevedibile che essa incida fortemente sia sulla compilazione del modello che sulla sua risoluzione. Tra la dimensione dell'input e la dimensione del modello può sussistere un legame diretto o indiretto. In OPPStandard il legame è diretto, in quanto i vincoli sono quantificati dagli insiemi di nodi ed archi, a prescindere dalla loro pinnabilità. In OPPRestricted ed OPPConservative il legame è indiretto, in quanto i vincoli sono quantificati dagli insiemi di nodi ed archi pinnabili. In questi due modelli è dunque il grado di pin a determinare la dimensione del modello: la dimensione dell'input vi stabilisce un limite superiore.

La distribuzione dell'input determina l'eterogeneità delle soluzioni ammissibili del modello. Pertanto essa incide fortemente sulla sua risoluzione. Ciò è vero in generale in quanto una maggiore eterogeneità permette sempre una maggiore efficienza nel discriminare la bontà delle soluzioni ammissibili. Questo principio è particolarmente vero nel nostro caso, in quanto adottiamo l'ottimizzatore IBM® MIP Optimizer con risolutore Simplex Solver [**?**], il quale implementa l'algoritmo del Simplesso [44].

4.2 Metrica

Le performance sono valutate adottando come metrica la mediana del tempo di esecuzione in millisecondi, valutata su un campione di 20 iterazioni. Abbiamo utilizzato la mediana, piuttosto che la media, in quanto risulta più robusta ai valori outlier registrabili durante le osservazioni [57]. Possono infatti verificarsi fluttuazioni nei tempi di esecuzione dovuti all'incapsulamento degli esperimenti nelle test-suite JUnit, alla liberazione della memoria[1] da una iterazione all'altra dell'esperimento, nonchè all'interazione della JNI con la libreria nativa CPlex. Consideriamo tali fluttuazioni come osservazioni anomale, in quanto una implementazione dei modelli libera dalla necessità di una prototipazione rapida ne eliminerebbe i fattori scatenanti.

4.3 Ambiente di esecuzione

L'applicazione OPMap è stata installata ed eseguita su un laptop HP® Pavilion Sleekbook 15-B030EL con processore a 64bit Intel® Core i5-3317U CPU @ 1.70GHz, memoria DDR3 da 4GB e hard-disk Seagate® Momentus SpinPoint M8 SATA 500GB @ 3Gbps. La macchina monta il sistema operativo Linux® Ubuntu 15.04, compilatore Oracle® Java 1.8, ambiente di test JUnit4.0 e libreria IBM® CPlex 12.6.2 Academic Edition.

[1] l'occupazione di memoria da parte delle strutture dati per la compilazione e risoluzione dei modelli è così ingente da rendere necessaria una liberazione esplicita al termine di ogni iterazione. Ciò è vero sia a livello Java Concert Technology che a livello CPLEX Core. Ad ogni iterazione dunque, la Java Virtual Machine (JVM) deve invocare il Java Garbage Collector (JGC), e al contempo la Java Native Interface (JNI) induce lo strato nativo ad uno uso massiccio delle free di sistema. A ciò si aggiungono le naturali invocazioni del JGC indotte dal framework JUnit.

4.4 Esperimenti e Risultati

Mostriamo e commentiamo ora i risultati ottenuti dagli esperimenti. Nel progetto OPMap e nel seguito adottiamo la seguente nomenclatura per gli esperimenti eseguiti:

[scopo-parametro]

dove lo scopo può essere la compilazione (C) o la risoluzione (R) del modello; mentre il parametro può essere il numero di nodi computazionali (EXNode), il umero di nodi operazionali (OPNode), il fattore di pinnabilità (PINFactor) o il fattore di eterogeneità (DIVFactor) sia dell'architettura che dell'applicazione. Ad esempio, l'esperimento [R-OPNode] ha lo scopo di valutare le performance della risoluzione del modello al variare del numero di nodi operazionali.

4.4.1 Esperimenti sulla compilazione

In questa suite di esperimenti mostriamo la sensibilità dei modelli in funzione della dimensione dell'input, durante la fase di compilazione. Lo scopo non è solo individuare quale sia il modello compilabile più rapidamente, bensì anche verificare se incida maggiormente sulle performance un aumento in dimensione dell'architettura o uno stesso aumento in dimensione dell'applicazione.

Esperimento [C-EXNode]/[C-OPNode] Con l'esperimento C-EXNode vogliamo osservare il comportamento dei modelli durante il processo di compilazione, in risposta all'aumento del numero di nodi computazionali. Con l'esperimento C-OPNode vogliamo invece osservare lo stesso comportamento, in risposta all'aumento del numero di nodi operazionali.

Confrontando i risultati degli esperimenti in Figura 4.1 e Figura 4.2, notiamo come tutti i modelli ottengano tempi minori ed una maggiore robustezza in corrispondenza di un aumento di nodi operazionali piuttosto che di un eguale aumento di nodi computazionali. In entrambi gli esperimenti, i modelli OPPStandard e OPPRestricted sono caratterizzati dagli stessi andamenti, mentre il modello OPPConservative se ne distacca sensibilmente,

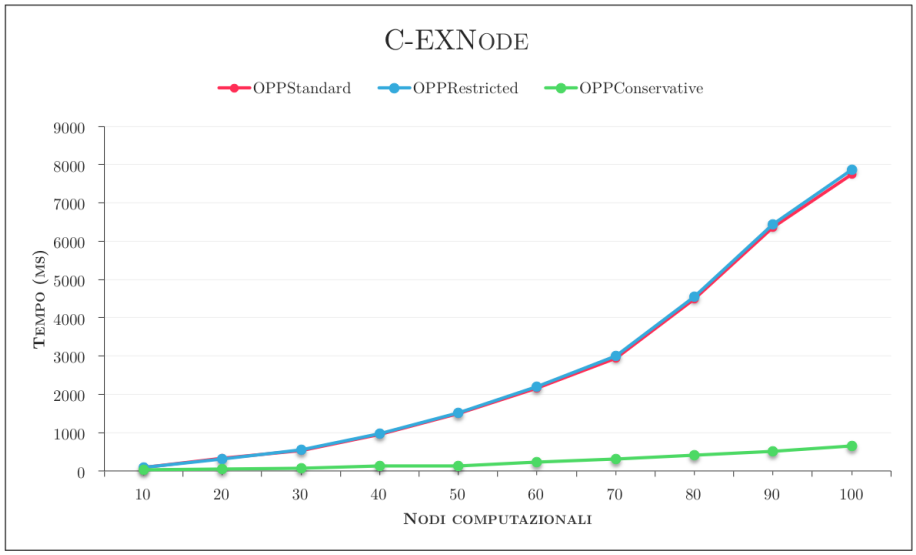


Figura 4.1: Risultati esperimento [C-EXNode].

dimostrando di poter ammortizzare meglio degli altri sia l'aumento dei nodi computazionali che di quelli operazionali. Possiamo dunque concludere che in fase di compilazione, ogni modello è più robusto all'aumento di nodi operazionali che ad un eguale aumento di nodi computazionali, e che i vincoli di connettività espressi con la formulazione della conservazione del flusso garantiscono una forte riduzione nella dimensione del modello.

Esperimento [C-PINFactor] Con l'esperimento C-PINFactor vogliamo osservare il comportamento dei modelli durante il processo di compilazione, in risposta all'aumento del fattore di pin, dunque del conseguente grado di pin di ogni nodo operazionale.

Osservando i risultati dell'esperimento in Figura 4.3, notiamo profonde differenze nelle curve in termini di valore iniziale ed elasticità. La curva OPPStandard ha un valore iniziale relativamente elevato, e rimane pressochè inelastica all'aumento del fattore di pin. La curva OPPRestricted ha un valore iniziale tedente a zero ed una apprezzabile elasticità. La curva OPPConservative ha un valore iniziale tendente a zero ed una elasticità molto ridotta. Possiamo dunque concludere che in fase di compilazione, l'aumento del fattore di pin incide sulle performance dei soli modelli OPPRestricted ed OPPConservative, mentre è pressochè ininfluente per il modello OPP-

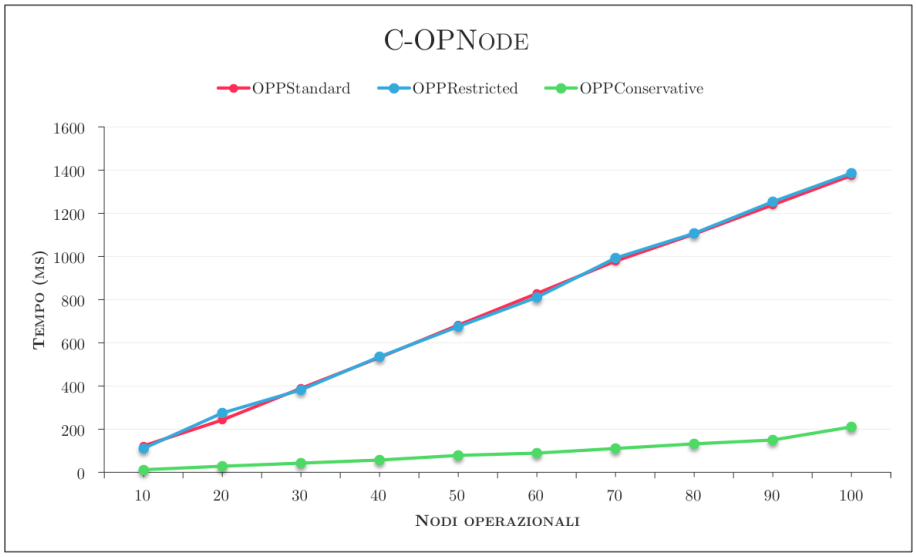


Figura 4.2: Risultati esperimento [C-OPNode]

Standard. Ciò è certamente dovuto alla restrizione del dominio. Inoltre deduciamo che i vincoli di connettività espressi con la formulazione della conservazione del flusso garantiscono una maggiore robustezza all'aumento del fattore di pin.

4.4.2 Esperimenti sulla risoluzione

In questa suite di esperimenti mostriamo la sensibilità dei modelli in funzione della dimensione e distribuzione dell'input, durante la fase di risoluzione. Lo scopo non è solo determinare il modello più performante, bensì anche verificare se incida maggiormente sulle performance un aumento in dimensione dell'architettura o uno stesso aumento in dimensione dell'applicazione.

Questi esperimenti mostrano risultati direttamente correlati a quelli ottenuti per la fase di compilazione, in quanto la dimensione del modello incide fortemente sui tempi di risoluzione, sia durante il preprocessamento che durante l'effettiva applicazione dell'algoritmo risolutivo.

Esperimento [R-EXNode]/[R-OPNode] Con l'esperimento R-EXNode vogliamo osservare il comportamento dei modelli durante il processo di risoluzione, in risposta all'aumento del numero di nodi computazionali. Con l'e-

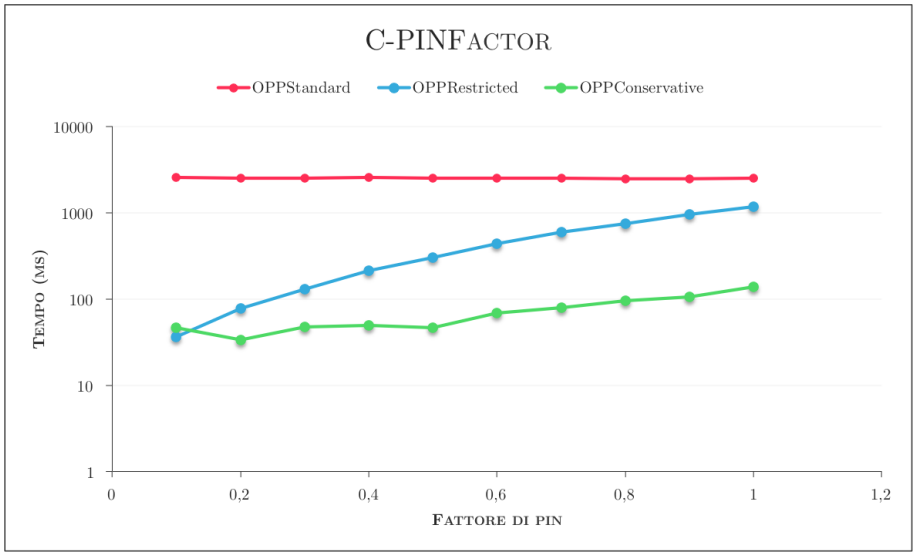


Figura 4.3: Risultati esperimento [C-PINFactor]

sperimento R-OPNode vogliamo invece osservare lo stesso comportamento, in risposta all'aumento del numero di nodi operazionali.

Confrontando i risultati degli esperimenti in Figura 4.4 e Figura 4.5, notiamo come tutti i modelli ottengano tempi minori ed una maggiore robustezza in corrispondenza di un aumento di nodi operazionali piuttosto che di un eguale aumento di nodi computazionali. Ancora una volta, gli andamenti dei modelli OPPStandard ed OPPRestricted risultano pressochè analoghi. Questa volta però riscontriamo una particolare esposizione dei due modelli a fluttuazioni imputabili alla gestione della memoria in corrispondenza dell'aumento dei nodi computazionali[2]. Il modello OPPConservative si mostra più efficiente degli altri, ma con una escursione minore rispetto a quella rilevata in fase di compilazione.

Esperimento [R-PINFactor] Con l'esperimento R-PINFactor vogliamo osservare il comportamento dei modelli durante il processo di risoluzione, in risposta all'aumento del fattore di pin, dunque del conseguente grado di pin di ogni nodo operazionale.

[2]esperimenti condotti più volte sui modelli OPPStandard e OPPRestricted con dimesioni di input maggiori di quelle riportate, mostrano picchi di latenza isolati analoghi a quello riscontrato in Figura 4.4.

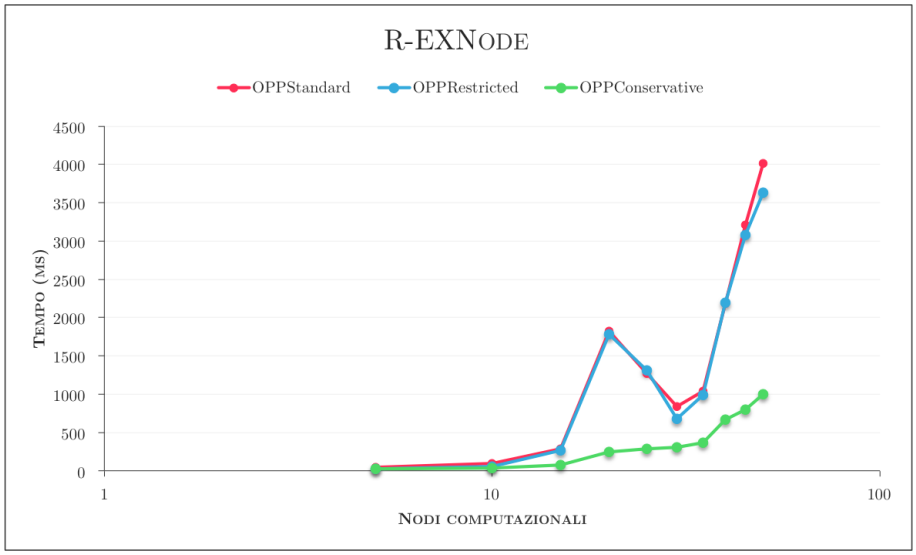


Figura 4.4: Risultati esperimento [R-EXNode]

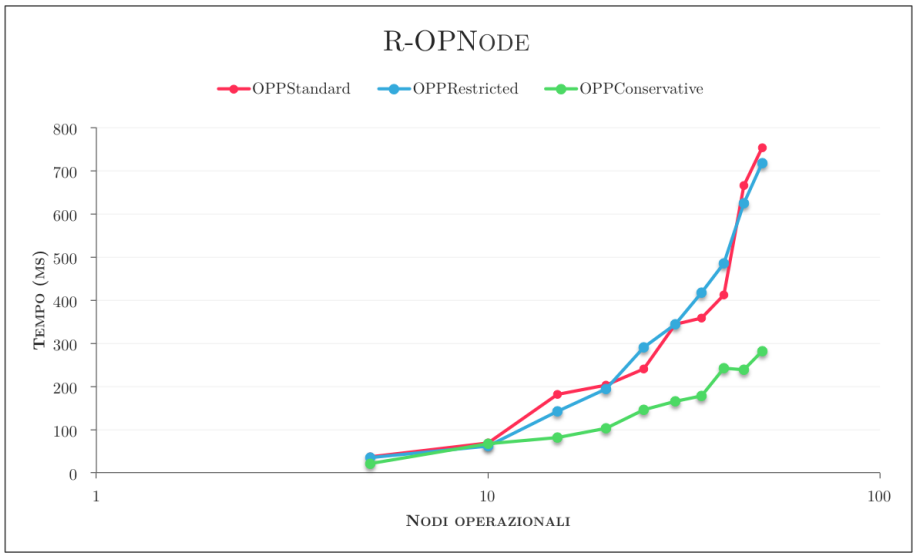


Figura 4.5: Risultati esperimento [R-OPNode]

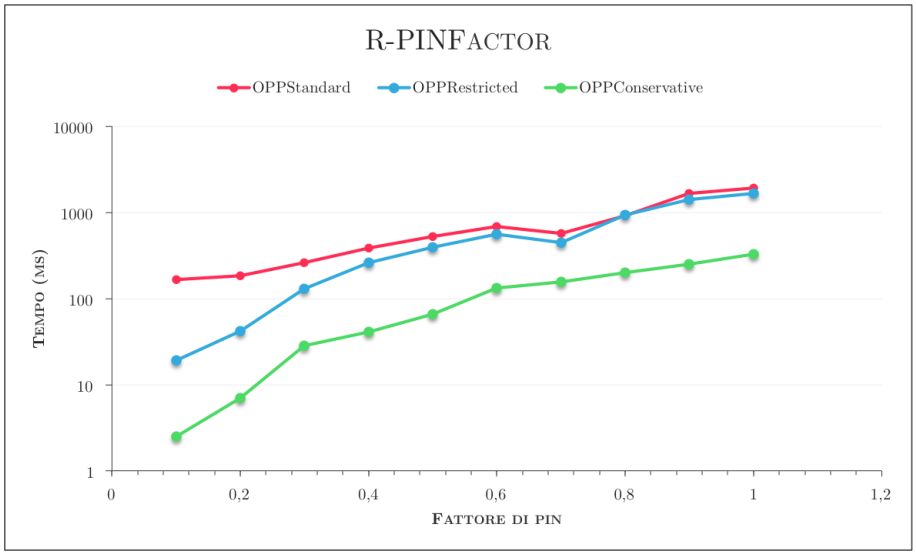


Figura 4.6: Risultati esperimento [R-PINFactor]

Osservando i risultati dell'esperimento in Figura 4.6, notiamo un comportamento profondamente diverso rispetto a quello riscontrato in C-PINFactor. Il modello OPPStandard non risulta più insesibile all'aumento del fattore di pin. Il modello OPPRestricted assume presto l'andamento di OPPStandard. Il modello OPPCoservative si mostra sempre più robusto degli altri due, ma meno di quanto risconntrato in fase di compilazione.

Esperimento [R-DIVFactor] Con l'esperimento R-DIVFactor vogliamo osservare il comportamento dei modelli durante il processo di risoluzione, in risposta all'aumento del fattore di eterogeneità dell'architettura e dell'applicazione.

Osservando i risultati dell'esperimento in Figura 4.7, notiamo una generale robustezza dei modelli al fattore di eterogeneità. Nei modelli OPPStandard ed OPPRestricted si può notare inoltre la predispozione a fluttuazioni dovute alla gestione della memoria[3]. Possiamo concludere che, escluse tali fluttuazioni, il fattori di eterogeneità ha poca influenza sulla performance in fase di risoluzione.

[3]esperimenti ripetuti sui modelli OPPStandard e OPPRestricted con dimesioni di input maggiori di quelle riportate, mostrano picchi di latenza isolati analoghi a quelli riscontrati in Figura 4.7.

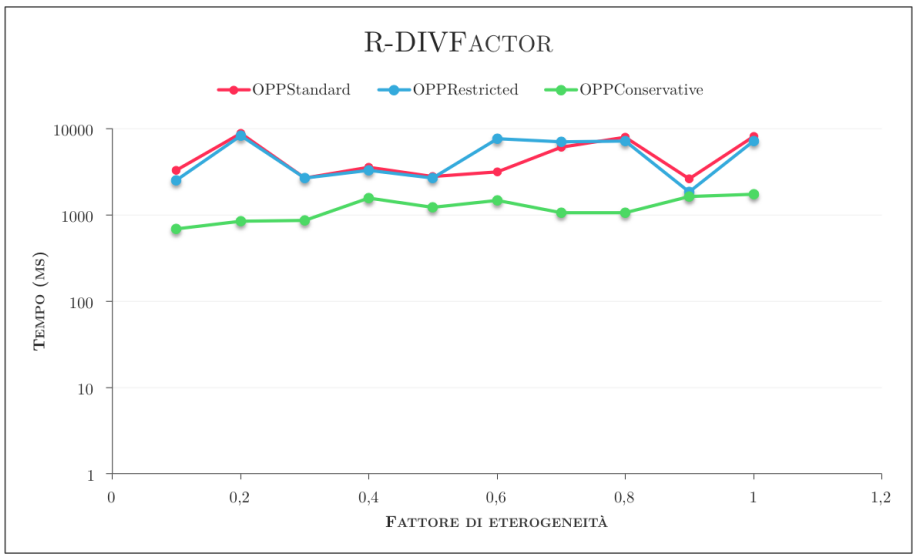


Figura 4.7: Risultati esperimento [R-DIVFactor]

Capitolo 5

Conclusioni

In questo lavoro di tesi è stato presentato il problema del posizionamento degli operatori per applicazioni di Data Stream Processing. Il problema è stato contestualizzato all'interno del fenomeno dei Big Data e dell'Information Flow Processing per far assaporare al lettore l'attuale valore dei modelli di ottimizzazione applicati al processamento dei big dataset.

Del problema di posizionamento sono state fornite tre distinte formulazioni: OPPStandard, OPPRestricted e OPPConservative. E' stata descritta l'implementazione dei tre modelli all'interno del progetto OPMap, il quale utilizza la libreria IBM CPlex per la loro compilazione e risoluzione.

Tali modelli sono stati poi sottoposti ad un'analisi sperimentale volta ad enfatizzarne le differenze prestazionali in funzione della dimensione e distribuzione dell'input. L'analisi sperimentale ha verificato le migliori performance della formulazione OPPConservative, sia in fase di compilazione del modello che durante quella della sua risoluzione. La formulazione in questione registra, rispetto alle altre, bassi tempi di esecuzione ed una comoda inelasticità alle fondamentali variazioni dell'input.

In futuro, speriamo che il progetto OPMap possa diventare un utile strumento per la prototipazione rapida di modelli OPP e che possa fornire strumenti di benchmark a supporto dello sviluppo di soluzioni euristiche al problema.

Bibliografia

[1] J. Anuradha, "A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology," *Procedia Computer Science*, vol. 48, pp. 319–324, 2015.

[2] VCloudNews, "Every day Big Data Statistics," 2015.

[3] J. Gantz, D. Reinsel, and B. D. Shadows, "The Digital Universe in 2020," *IDC iView "Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East"*, vol. 2007, no. December 2012, pp. 1–16, 2012.

[4] "State of the Internet," tech. rep., Akamai, 2015.

[5] J. Gantz, "The Expanding Digital Universe," tech. rep., IDC, 2007.

[6] "Big Data papers trends," 2015.

[7] "'Big Data' and 'Apache Hadoop'," tech. rep., Google Trends.

[8] C. M. Christensen, *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. Harvard Business Press, 2013.

[9] J. Kelly, D. Floyer, D. Vellante, and S. Miniman, "Big Data Vendor Revenue And Market Forecast 2012-2017," *Wikibon*, no. March, p. 1, 2015.

[10] B. Marr, *Big Data Case Study Collection*. Wiley, 1 ed., 2015.

[11] C. Bange, T. Grosser, and N. Janoschek, "Big Data Use Cases," tech. rep., BARC, 2015.

[12] "Disk Drive Prices (1955-2015)," tech. rep., JCMIT, 2015.

[13] "Compound Annual Growth Rate (CAGR)."

[14] A. Nadkarni, F. Iris, and D. Laura, "Worldwide Storage in Big Data Forecast, 2015-2019," tech. rep., IDC, 2015.

[15] M. Schroeck, R. Shockley, J. Smart, D. Romero-Morales, and P. Tufano, "Analytics: The real-world use of big data," *IBM Global Business Services Saïd Business School at the University of Oxford*, pp. 1–20, 2012.

[16] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. a.S. Netto, and R. Buyya, "Big Data computing and clouds: Trends and future directions," *Journal of Parallel and Distributed Computing*, vol. 79-80, pp. 3–15, 2014.

[17] J. J. Berman, "Introduction," *Principles of Big Data*, pp. xix–xxvi, 2013.

[18] S. Suthaharan, "Big Data Classification: Problems and Challenges in Network Intrusion Prediction with Machine Learning," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, pp. 70–73, apr 2014.

[19] G. Cugola and A. Margara, "Processing Flows of Information: From Data Stream to Complex Event Processing," *ACM Comput. Surv.*, vol. 44, no. i, pp. 15:1–15:62, 2012.

[20] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[21] T. Universität and D. Dresden, "DEBS Grand Challenge : Real Time Data Analysis of Taxi Rides using StreamMine3G," pp. 269–276.

[22] S. Schneider, M. Hirzel, and B. Gedik, "Tutorial: Stream Processing Optimizations," in *Proceedings of the 7th ACM international conference on Distributed event-based systems - DEBS '13*, DEBS '13, (New York, New York, USA), p. 249, ACM Press, 2013.

[23] Google, "Hot Searches Trends," 2015.

[24] Twitter, "Streamig API," 2015.

[25] "The Large Hadron Collider," 2015.

[26] CERN, "Animation shows LHC data processing," 2015.

[27] CERN, "The Worldwide LHC Computing Grid," 2015.

[28] Z. Jerzak, T. Heinze, M. Fehr, D. Gröber, R. Hartung, and N. Stojanovic, "The DEBS 2012 grand challenge," in *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems - DEBS '12*, (New York, New York, USA), pp. 393–398, ACM, ACM Press, 2012.

[29] C. Mutschler, H. Ziekow, and Z. Jerzak, "The DEBS 2013 grand challenge," in *Proceedings of the 7th ACM international conference on Distributed event-based systems - DEBS '13*, (New York, New York, USA), p. 289, ACM, ACM Press, 2013.

[30] Z. Jerzak and H. Ziekow, "The DEBS 2014 grand challenge," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems - DEBS '14*, (New York, New York, USA), pp. 266–269, ACM, ACM Press, 2014.

[31] Z. Jerzak and H. Ziekow, "The DEBS 2015 grand challenge," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems - DEBS '15*, (New York, New York, USA), pp. 266–268, ACM, ACM Press, 2015.

[32] S. A. Cook, "An overview of computational complexity," *Communications of the ACM*, vol. 26, pp. 400–408, jun 1983.

[33] L. Fortnow and S. Homer, "A Short History of Computational Complexity," *Science*, pp. 1–26, 2002.

[34] M. W. Krentel, "The complexity of optimization problems," *Journal of Computer and System Sciences*, vol. 36, pp. 490–509, jun 1988.

[35] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, vol. 33. Springer-Verlag, 2004.

[36] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, 1 ed., 1990.

[37] S. Martello and P. Toth, "Algorithms for Knapsack Problems," *North-Holland Mathematics Studies*, vol. 132, no. C, pp. 213–257, 1987.

[38] P. Serafini, *Ricerca Operativa*. Udine, Italy: Springer Verlag, 1 ed., 2009.

[39] S. Martello, G. Laporte, M. Minoux, and C. Ribiero, *Surveys in Combinatorial Optimization*, vol. 31. Elsevier, 1987.

[40] J. Kleinberg and E. Tardos, "Approximation algorithms for classification problems with pairwise relationships: metric labeling and Markov random fields," *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, vol. 49, no. 5, pp. 616–639, 1999.

[41] J. Chuzhoy and J. Naor, "The hardness of metric labeling," *45th Annual IEEE Symposium on Foundations of Computer Science*, vol. 36, no. 2, pp. 498–508, 2004.

[42] D. M. Curry and C. H. Dagli, "Computational Complexity Measures for Many-objective Optimization Problems," *Procedia Computer Science*, vol. 36, pp. 185–191, 2014.

[43] IBM, "IBM's ILOG acquisition," 2009.

[44] G. B. Dantzig, A. Ordern, and P. Wolfe, "The Generalized Simplex Method," *Pacific Journal of Mathematics*, vol. 5, no. 2, pp. 183–195, 1955.

[45] IBM, "IBM ILOG CPLEX Optimization Studio," 2015.

[46] IBM, "IBM CPLEX Optimizers Performance," 2015.

[47] IBM, "CPLEX User's Manual," 2014.

[48] IBM, "IBM Optimization Programming Language (OPL)," 2015.

[49] IBM, "Mathematical Programming vs. Constraint Programming," 2015.

[50] IBM, "Getting Started with the CPLEX Studio IDE," 2014.

[51] IBM, "OPL Language User's Manual," 2014.

[52] IBM, "OPL Language Reference Manual," 2014.

[53] G. Marciani, "OPMap-OPL on Github," 2015.

[54] Oracle, “Oracle Java Native Interface,” 2015.

[55] JUnit, “JUnit,” 2015.

[56] G. Marciani, “OPMap on Github,” 2015.

[57] S. M. Ross and F. Morandin, *Probabilità e statistica per l’ingegneria e le scienze*. 2015.