SAPIENZA UNIVERSITÀ DI ROMA

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

MODELLAZIONE E VALUTAZIONE DI DBMS RELAZIONALI BASATI SU LOCK E PATTERN DI ACCESSO NON UNIFORMI

RELATORE:
PROF. BRUNO CICIANI

LAUREANDO:
ROBERTO PALMIERI

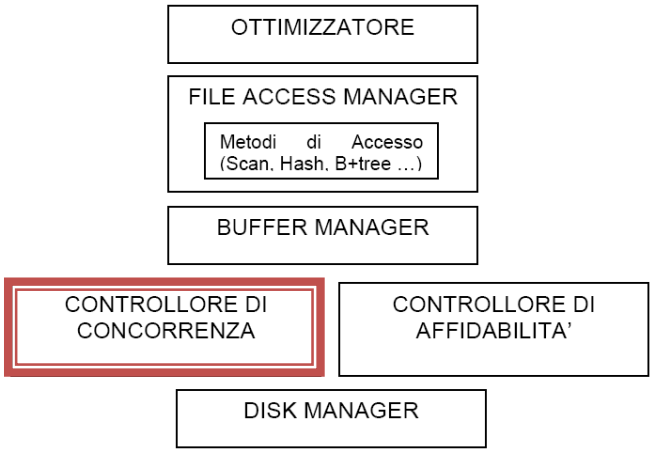
ANNO ACCADEMICO 2007/2008

INIDICE DEI CONTENUTI

Introduzione

Tutti i sistemi attualmente presenti e responsabili della gestione di molti aspetti della produttività hanno come supporto informatico un sistema transazionale. Per caratterizzare tali sistemi, definiamo a grosse linee una transazione come un'unità elementare di lavoro svolta da un'applicazione cui si vogliono associare particolari caratteristiche di correttezza, robustezza e isolamento.
In questa ottica un sistema transazionale è un sistema che mette a disposizione un meccanismo per la definizione e l'esecuzione delle transazioni.

I sistemi transazionali sono stati da sempre oggetto di studio ed in questo documento ci restringiamo ad una classe preponderante dei sistemi transazionali, ovvero i DBMS (Data Base Management System). Questi rappresentano un contenitore di dati ottimizzato per la gestione di accessi concorrenti e di ricerche (query). Le componenti fondamentali di un DBMS sono riportate in figura.



Il *controllo di concorrenza* è quell'attività svolta dal DBMS che consente l'esecuzione contemporanea di più transazioni sulla medesima base di dati, senza interferenze reciproche. Esso viene assicurato da diversi moduli del sistema e sinteticamente rappresentati in figura nel *Controllore di Concorrenza*.

In ogni DBMS il "Controllore di Concorrenza" implementale regole con le quali schedulare le operazioni delle varie transazioni che arrivano al sistema. Per far ciò vengono usate delle strategie o protocolli che, pur mantenendo tutte le proprietà di isolamento tra le transazioni, prediligono alcune politiche piuttosto che altre quindi favorendo determinati scenari di utilizzo e sfavorendole altri.
Queste strategie si possono classificare in tre tipi: basate su locking (o pessimistiche), ottimistiche e multiversione. Ogni strategia può essere implementata da un protocollo ma esistono protocolli che implementano alcuni aspetti di più strategie, questo per compensare agli svantaggi di una singola strategia.

In questo contesto articolato e di difficile analisi si inseriscono i metodi di valutazione delle performance di un sistema transazionale. Tra i metodi di valutazione vi sono quelli basati su modelli, particolarmente adatti alla progettazione di sistemi transazionali, che posso essere di tipo analitico o simulativo.

- I modelli analitici sono caratterizzati da una certa flessibilità e da un basso costo di implementazione ma presentano notevoli limitazioni nelle condizioni in cui il fenomeno da analizzare diventa particolarmente complesso. Proprio per questo

vengono largamente utilizzati per modellare fenomeni in cui vengono applicate una serie di esemplificazioni. La controindicazione ricade in una possibile perdita di affidabilità qualora i comportamenti risultassero troppo complessi da analizzare.

- I modelli simulativi riescono meglio a catturare le situazioni di particolare complessità ed in cui è presente il contributo di più effetti separati, ma in genere hanno costi di realizzazione più alti e necessitano di una potenza di elaborazione elevata.

Nello specifico in questo lavoro verranno analizzati i protocolli basati su strategie di locking ed in particolare sul protocollo del 2PL (Two Phase Locking).

In letteratura questa famiglia di protocolli basata su locking è stata molto trattata in vari studi, ma in questo lavoro la modellazione di tali protocolli verrà specializzata e quindi approfondita nella caratterizzazione del carico di lavoro (workload) del DBMS. Il carico di lavoro è un insieme di parametri che descrivono le condizioni di funzionamento di un sistema. In letteratura non è attualmente presente un lavoro che considera come parametro fondamentale del carico di lavoro il pattern di accesso ai dati da parte delle transazioni. In questo lavoro si cercherà di dimostrare come la caratterizzazione della modalità con la quale le transazioni accedono ai dati influenza pesantemente i tempi di risposta del sistema e quindi un degrado delle prestazioni. Questo tipo di caratterizzazione del carico è già presente all'interno di alcuni benchmark di riferimento per sistemi transazionali come TCP-C, quindi un possibile terreno fertile per l'applicazione del modello studiato nei capitoli successivi.

Il protocollo scelto, ovvero il 2PL, è uno dei protocolli maggiormente utilizzati dai DBMS commerciali, per questo è stato preso in esame. La caratterizzazione del workload si ricondurrà nello specifico nella modellazione di pattern di accesso ai dati da parte delle varie transazioni. Gli effetti sulle prestazioni dovuti a tali pattern di accesso si manifestano principalmente attraverso incrementi dei tempi di risposta delle transazioni e riduzioni del throughput del sistema, che sono parametri fondamentali per la valutazione delle prestazioni di un sistema transazionale.

Attraverso la modellazione analitica verrà formalizzato proprio questo aspetto, ovvero la caratterizzazione dei pattern di accesso ai dati nei DBMS basati su protocollo di concorrenza di tipo 2PL.

Capitolo 1
Strategie per il controllo della concorrenza nei sistemi transazionali

Come indicato anche nel paragrafo introduttivo, nei sistemi transazionali riveste un ruolo fondamentale il protocollo di concorrenza adottato. La necessità di mantenere un alto livello di performance e contemporaneamente di garantire atomicità e determinati livelli di isolamento ha portato alla definizione di diverse strategie per il controllo della concorrenza nei DBMS.
In questo capitolo verrà presentata una carrellata delle varie metodologie con cui ci si approccia al problema dello sviluppo di un protocollo di concorrenza.

1.1.1 Classificazione dei protocolli per il controllo della concorrenza

I protocolli di concorrenza per DBMS possono essere classificati e raggruppati attraverso queste tre classi:

- Locking;
- Strategie Ottimistiche;
- Strategie Pessimistiche;

A causa della varietà dei problemi da affrontare con i protocolli per il controllo della concorrenza e delle diverse tecniche adottate per risolverli non è possibile definirne in tutti i casi una netta suddivisione tra le classi sopra definite. Le diverse tecniche spesso vengono utilizzate congiuntamente per ottenere controlli che meglio si adattano alle diverse situazioni.

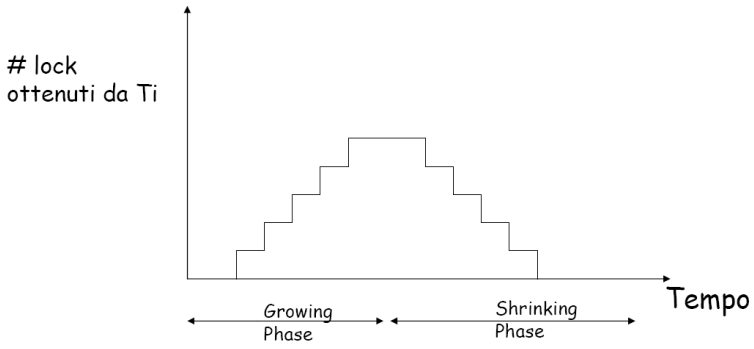
1.1.2 Locking

La strategia basata su locking è largamente la più utilizzata nei controlli di concorrenza dei sistemi transazionali. I protocolli basati sui locks risultano anche i più analizzati nei lavori disponibili in letteratura scientifica. L'idea di base è che ogni dataitem è associato ad un lock.
Una transazione che vuole accedere ad un dataitem deve preventivamente effettuare una richiesta di lock. Se il lock relativo al dataitem non è mantenuto da un'altra transazione allora gli viene concesso, altrimenti la transazione entra in uno stato di attesa sino al rilascio del lock. In tal modo una transazione può effettuare un accesso ad un dataitem solo quando non vi è alcun lock da parte di altre transazioni. Per consentire accessi concorrenti sia in scrittura che in lettura su uno stesso dataitem si utilizzano due tipi di lock: shared ed exclusive. Un lock shared viene chiesto da una transazione per un'operazione di lettura e viene concesso solo se non vi è un lock exclusive sullo stesso dataitem. In tal modo due transazioni posso leggere contemporaneamente lo stesso dataitem. Un lock exclusive è richiesto per effettuare un'operazione di scrittura e viene concesso solo se non vi è un altro tipo di lock sullo stesso dataitem in possesso di un'altra transazione. In tal modo una scrittura può avvenire solo se non vi sono altre letture o scritture in corso sullo stesso dataitem. I lock possono avere granularità diversa, ossia essere associati sia ad un dataitem che a set di dataitems.
Generalmente questi set di dataitems possono corrispondere a tuple, blocchi o tabelle. L'utilizzo dei lock può portare al verificarsi di deadlocks. Le tecniche utilizzate per far fronte a queste situazioni sono basate sul riconoscimento del deadlock attraverso grafi di attesa delle transazioni e sull'utilizzo di timeouts. In entrambi i casi è necessario abortire una transazione per assicurarsi che il deadlock sia risolto. Una tecnica alternativa consente di prevenire i deadlock attraverso l'uso di priorità.
L'abort di una transazione può generare il fenomeno del cascading rollback. Ossia le transazioni che hanno letto dati scritti dalla transazione che effettua l'abort devono essere a loro volta abortite. Un'altro problema dovuto agli abort è la possibile scrittura di dati che

dipendono da altri dati scritti e poi annullati a causa di abort. Per garantire l'assenza di tali fenomeni è necessario che le esecuzioni delle transazioni garantiscano la recuperabilità, ossia che nessuna transazione effettui il commit a meno che non sia stato effettuato da tutte le transazioni da cui ha letto.

Uno dei protocolli più diffusi basati sul locking è il *two-phase locking* (2PL).

Questo protocollo utilizza locks exclusive e shared e garantisce la serializzabilità. La regola di base del 2PL consiste nell'effettuare tutte le richieste di lock prima che ne venga effettuato un rilascio. In figura si può osservare la curva tipica di richiesta dei lock del 2PL e nello specifico le due fasi del protocollo:



Una variante del 2PL, detta Conservative 2PL, consente di evitare i deadlocks. Ciò è ottenuto richiedendo che ogni transazione effettui una pre-dichiarazione dei set di dataitems che vuole accedere in scrittura ed in lettura. La variante più diffusa del 2PL e detta *strict 2PL*. Questa versione prevede il rilascio dei locks ottenuti da una transazione solo dopo l'operazione di commit od abort. Un tale comportamento può essere causa di tempi di attesa molto lunghi da parte delle transazioni bloccate dalla presenza dei locks, ma consente di evitare il cascading rollback e assicura recuperabilità. Il 2PL è utilizzato anche in contesti distribuiti. Tale implementazione prende il nome di *distribuited-2PL*.

1.1.3 Strategie ottimistiche

La caratteristica peculiare delle strategie ottimistiche è che non bloccano le transazioni. Le anomalie dovute agli accessi concorrenti sono evitate effettuando l'abort delle transazioni. Vi sono diverse varianti dei protocolli che utilizzano tali tecniche. Tra queste il *timestamp ordering*. Questo protocollo prevede l'assegnazione ad ogni transazione di un timestamp univoco che ne rappresenta il tempo logico. La regola di base del timestamp ordering consente l'esecuzione di un'operazione p, appartenente alla transazione T_1, prima di un'operazione q, appartenente alla transazione T_2, e con la quale risulta in conflitto, se e solo se il timestamp di T_1 risulta inferiore al timestamp di T_2. Questa regola è sufficiente a garantire esecuzioni serializzabili.

Una semplice implementazione del timestamp ordering è il *basic timestamp ordering*. Questo protocollo non fa altro che abortire le transazioni che richiedono operazioni in ritardo rispetto all'istante massimo consentito dalla regola del timestamp ordering.

Una distinzione tra i protocolli ottimistici può essere effettuata a seconda del momento in cui una transazione da abortire viene effettivamente abortita. In questo caso un protocollo che per effettuare l'abort attende che la transazione esegua tutte le operazioni previste viene detto ottimistico *puro*. Un protocollo che effettua l'abort in concomitanza ad un accesso della transazione ad un dataitem non più valido viene identificato come ottimistico di tipo *broadcast*.

1.1.4 Tecniche multiversione

Le tecniche multiversione nascono con l'obbiettivo di evitare le attese o gli abort causati dalle operazioni di lettura. Nei protocolli che utilizzano le versioni multiple in ogni istante un dataitem può essere presente in più versioni. Le versioni vengono create ogni volta che una transazione effettua un'operazione di scrittura. In tal modo una nuova scrittura non elimina le versioni dei dataitem presenti nella base di dati prima di tale operazione. La presenza di più versioni consente di eseguire tutte le operazioni di lettura senza che siano causa di attesa o di abort. Le differenti versioni di ogni dataitem vengono identificate attraverso alcuni timestamps relativi alle transazioni che le creano. Attraverso questi timestamps è possibile selezionare di volta in volta la versioni da leggere in modo da fornire alle transazioni una vista consistente della base di dati.

Le tecniche multiversione sono utilizzate congiuntamente sia a strategie di tipo ottimistico sia basati su locking. Il protocollo *multivesion timestamp ordering* utilizzali la tecnica dei timestamps vista in precedenza e le versioni multiple dei dati, evitando di abortire le transazioni che effettuano letture in ritardo. Il protocollo *two version 2PL* utilizza invece al massimo due versioni per ogni dataitem ed i locks per risolvere i conflitti tra scritture.
Quando una transazione effettua una scrittura ottiene un lock exclusive sul dataitem. Le letture sullo stesso dataitem vengono effettuate utilizzando la precedente versione.

Capitolo 2

Modellazione e valutazione dei controlli di concorrenza

In questo capitolo si mettono in evidenza gli aspetti fondamentali che caratterizzano i processi di valutazione della performance dei sistemi transazionali. Si presentano i metodi, gli strumenti ed i modelli più largamente utilizzati ed i livelli di astrazione di questi ultimi. In particolare si prendono in considerazione i modelli analitici, le motivazioni che portano alla relativa costruzione ed i principali vantaggi e svantaggi di questi rispetto agli altri approcci. Successivamente il quadro si restringe alla modellazione dei protocolli di concorrenza dei sistemi transazionali, per poi passare ad una visione d'insieme dei modelli analitici e dei risultati presentati in letteratura scientifica.

2.1 La valutazione della performance dei sistemi transazionali

La performance di un sistema di elaborazione identifica la capacità di compiere una determinata quantità di lavoro in un dato intervallo di tempo. Il processo di valutazione della performance non viene effettuato considerando i guasti (*faults*) a cui il sistema in esame può essere soggetto. La proprietà di un sistema di mantenere integro il suo funzionamento, più

precisamente la probabilità, in funzione del tempo, che non si manifestino guasti, è definita affidabilità (*reliability*). La presenza di guasti influisce sulla quantità di tempo durante il quale il sistema può funzionare correttamente. Per misurare quest'ultima proprietà si definisce la disponibilità (*availability*). La disponibilità rappresenta la probabilità che un sistema non mostri malfunzionamenti in un determinato istante di tempo. Quest'ultima proprietà si differenzia dall'affidabilità proprio perchè è riferita ad uno specifico istante di tempo. Per effettuare una valutazione delle prestazioni che un sistema può effettivamente fornire si definisce la cosiddetta *performability* [3]. Sempre in riferimento ai sistemi di elaborazione, con il termine si esprime la capacità di un sistema di compiere una determinata quantità di lavoro considerando la possibilità del manifestarsi di malfunzionamenti. Si deduce che la performability dipende dalla performance di un sistema ed è inoltre legata alla relativa affidabilità e disponibilità.

La valutazione delle prestazioni di un sistema di elaborazione viene effettuata attraverso delle metriche di valutazione, o indici di prestazioni, che ne misurano l'efficienza nello svolgimento delle funzioni cui è preposto. In generale si utilizzano metodi basati su misurazione o metodi basati su modelli. I primi prevedono la valutazione degli indici di prestazione direttamente attraverso misure effettuate sul sistema reale, o su un prototipo di esso. I secondi prevedono l'utilizzo di strumenti simulativi od analitici che rispettivamente riproducono e modellano il comportamento del sistema reale. Un elemento importante, la cui caratterizzazione è fondamentale nel processo di valutazione della performance, è costituito dal carico di lavoro (*workload*) a cui è sottoposto un sistema di elaborazione.

Questi ed altri aspetti saranno analizzati nei successivi paragrafi in riferimento ai sistemi transazionali.

2.1.1 Metriche di valutazione

La performance di un sistema transazionale può essere valutata dal punto di vista dell'utente o del sistema. Nel primo caso la metrica di riferimento è il tempo di risposta (*response time*) di una transazione, ossia il valore che ne misura la durata totale. Il response time dipende dai tempi di servizio (*service time*), dai tempi di attesa (*waiting time*), dai ritardi introdotti dalla rete (*network delay*) e dai tempi che intercorrono tra le risposte del sistema e le successive richieste di operazioni dell'utente (*think time)*. Il tempo di risposta medio (*mean response*

time) è ottenuto mediando i tempi di risposta rilevati durante un determinato intervallo di tempo. Dal punto di vista del sistema la metrica più significativa è il *throughput*, che è dato dal numero di transazioni che vengono eseguite nell'unità di tempo (TPS). Il throughput viene calcolato come rapporto tra il numero di transazioni eseguite in un dato intervallo di tempo e la lunghezza dell'intervallo stesso. Il throughput può essere misurato anche considerando altri tipi di interazioni. Ad esempio il benchmark TPC-W [6] [7], finalizzato alla valutazione di sistemi per il commercio elettronico, definisce come metrica il WIPS (Web Interaction Per Second), dato dal numero di interazioni effettuate via web dall'utente nell'unità di tempo. Il throughput ed il mean response time dipendono da una serie di fattori per i quali è possibile definire ulteriori metriche. Il *numero di transazioni attive* nel sistema definisce, in relazione ad un determinato istante di tempo, il numero di transazioni già iniziate che devono ancora terminare. Tali transazioni quindi costituiscono parte del carico di elaborazione del sistema in relazione a tale istante. Per misurare lo sfruttamento di una risorsa di sistema (cpu, memoria, disco,...) si definisce l'*utilizzazione*, che misura la frazione di tempo medio per la quale la risorsa viene utilizzata. Quando una richieste viene ricevuta da una risorsa che risulta occupata a servirne altre, la nuova richiesta viene inserita in una coda in attesa che la risorsa si liberi. Il numero di richieste in attesa di una risorsa ne definisce la lunghezza della relativa coda . Le code possono avere dimensione finita, quindi le richieste che arrivano quando una coda è piena vengono scartate. Il rapporto tra il numero di richieste che vengono scartate ed il numero totali di richieste esprime la probabilità di perdita (*loss probability*) delle richieste.
Le metriche di cui si è parlato sono alcune delle più comunemente utilizzate nei processi di valutazione dei sistemi transazionali. A seconda delle caratteristiche del sistema in esame o dei relativi componenti è possibile individuare altri fattori che possono essere oggetto di valutazione, e quindi definirne metriche specifiche.

2.1.2 Metodi di valutazione

I metodi per la valutazione delle prestazioni di un sistema di elaborazione possono essere classificati in [14]:

- metodi basati su misurazione;
- metodi basati su modelli.

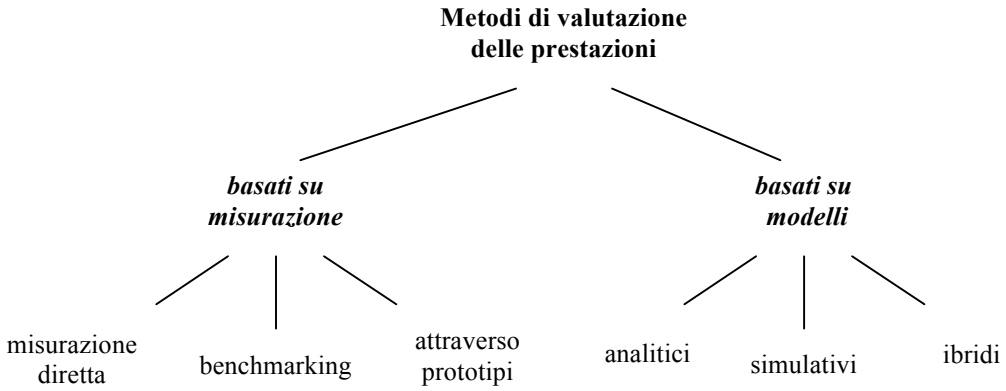
I metodi basati su misurazione si distinguono in:

- misurazione diretta;
- benckmarking;
- basati su prototipo;

La misurazione diretta consiste nel valutare gli indici di prestazione 'sul campo', mentre il sistema è in opera. Questo approccio consente di ottenere risultati molto affidabili. Uno svantaggio è legato al fatto che i risultati sono legati al carico di lavoro (*workload*) a cui il sistema è sottoposto durante la misurazione, quindi non sempre è possibile valutarne le prestazioni in relazione ad un workload con determinate caratteristiche. Un altro approccio è costituito dal *benckmarking*. Un *benckmark* definisce uno o più test a cui deve essere sottoposto un sistema ai fini di valutarne le prestazioni. Per tali test sono specificate le condizioni ed i requisiti che il sistema in analisi deve rispettare. In particolare un benckmark stabilisce le caratteristiche del workload a cui deve essere sottoposto il sistema. Il workload viene generato artificialmente, in genere attraverso strumenti software che emulano il comportamento degli utenti. I benckmarks costituiscono un valido strumento di confronto tra le prestazioni dei sistemi di elaborazione. In relazione ai sistemi transazionali alcuni dei benckmarks più utilizzati sono stati definiti dal TPC (Transaction Processing Performance Council) [12], consorzio che coinvolge i maggiori produttori di tali sistemi. Ad esempio il *TPC Benchmark™ App* (*TPC-App*) [13] è finalizzato al benchmarking di application servers che forniscono servizi web. Il workload è generato attraverso un software che emula il carico a cui è sottoposto un application server transazionale business-to-business che opera 24 ore su 24 , 7 giorni su 7.

La prototipazione è il terzo tipo di approccio basato su misurazione. In questo caso un prototipo fedele al sistema originale viene costruito per riprodurne il comportamento. L'utilizzo di un prototipo si rivela necessario quando il sistema sotto valutazione non è disponibile, ad esempio perchè in fase di progettazione.

I metodi basati su modelli sono orientati alla valutazione delle prestazioni attraverso una rappresentazione astratta del sistema in esame, dalla quale si possono ottenere informazioni sugli indici di prestazione secondo determinati livelli di accuratezza. Questi modelli sono detti astratti proprio perchè sono costruiti con determinati livelli di astrazione, quindi considerando solo gli aspetti necessari e/o sufficienti per lo scopo per cui sono realizzati.



Metodi di valutazione delle prestazioni dei sistemi di elaborazione

I modelli si possono distinguere in:

- modelli analitici;
- modelli simulativi;
- modelli ibridi;

I modelli analitici descrivono il comportamento del sistema attraverso variabili, parametri e relazioni matematiche tra essi. I valori degli indici di prestazione si ottengono attraverso la relativa risoluzione numerica. La valutazione di sistemi complessi può essere realizzata attraverso l'utilizzo di più modelli analitici, anche preesistenti, che rappresentano le diverse componenti del sistema, definendone le relazioni tra essi e risolvendo il modello generale ottenuto. Il modello realizzato in questo lavoro di tesi è di tipo analitico, per cui i metodi basati su tali strumenti saranno in seguito ampiamente oggetto del discorso.

I modelli simulativi prevedono la riproduzione del comportamento del sistema attraverso l'utilizzo di altri strumenti. Il software risulta particolarmente adatto a simulare un sistema transazionale, comprese le relative componenti hardware ed il comportamento dei relativi utenti. Uno dei vantaggi della simulazione software è, come per i modelli analitici, la possibilità di riutilizzare parti già realizzate per la simulazione dei componenti di un sistema.

Infine è possibile realizzare anche modelli ibridi utilizzando i modelli analitici per alcuni scopi e quelli simulativi per altri. Ad esempio per alcuni componenti di un sistema potrebbe risultare più conveniente l'utilizzo di un modello analitico per la relativa flessibilità ed il

basso costo. In altri casi potrebbe rivelarsi necessario lo sviluppo di un modello simulativo per catturare comportamenti specifici difficilmente modellabili attraverso strumenti analitici. Con un approccio misto occorre interfacciare correttamente i due tipi di modelli, quindi stabilirne gli inputs e gli outputs che consentono di riutilizzare i risultati analitici con i modelli simulativi e viceversa.

Quando viene realizzato un modello analitico un approccio largamente utilizzato per la verificata della validità è costituito dal confronto dei risultati numerici ottenuti con quelli ottenuti attraverso un modello simulativo. Il modello analitico realizzato in questo lavoro è stato validato secondo questo approccio, ossia utilizzando un modello simulativo che riproduce, con un determinato livello di astrazione, il comportamento di un sistema transazionale reale.

2.2 Modelli e strumenti analitici per la valutazione della performance dei sistemi transazionali

2.2.1 Principali motivazioni, vantaggi e svantaggi dei modelli analitici

L'utilizzo di metodi di valutazione basati su modelli analitici è legato a diverse motivazioni e comporta vantaggi e svantaggi rispetto agli altri approcci. Fondamentalmente la flessibilità ed il basso costo li rendono strumenti particolarmente adatti all'analisi del comportamento dei sistemi già in fase di progettazione. Essendo facilmente modificabili consentono di valutare e confrontare preventivamente diverse alternative, permettendo di stabilire il livello di astrazione al quale mantenersi. Un'altro vantaggio, legato in generale all'utilizzo dei modelli, è la possibilità di effettuare valutazioni indipendenti dei singoli componenti di un sistema, o dei sistemi stessi. Nella realtà spesso non è possibile far funzionare solo alcuni componenti di un sistema, od anche alcuni sistemi, indipendentemente da altri. Ad esempio in un sistema transazionale il modulo per il controllo della concorrenza non potrebbe funzionare senza un modulo di gestione delle transazioni. Ancora un DBMS (*DataBase Management System*) non potrebbe funzionare su un calcolatore senza la presenza di un sistema operativo. Dunque, le misure effettuate relativamente ai singoli componenti od ai singoli sistemi possono essere

spesso inevitabilmente influenzate dal comportamento degli altri. Al contrario, è possibile costruire modelli relativi solo ai componenti od ai sistemi di interesse, astraendo da tutto il resto.

I modelli analitici non necessitano di particolari strumenti se non di un calcolatore per la risoluzione numerica del modello. Rispetto ai metodi basati su misurazione non occorre il sistema reale od un prototipo. Nel caso dei modelli simulativi, la realizzazione di simulatori talvolta può essere alquanto onerosa. Inoltre essi potrebbero richiedere calcolatori potenti e numerosi test da effettuare per ottenere stime accurate ed intervalli di confidenza accettabili. C'è da considerare inoltre che ogni singolo test potrebbe richiedere tempi piuttosto lunghi. Tra i vantaggi dei modelli simulativi rispetto ai modelli analitici c'è la possibilità di testare il sistema con carichi di lavoro anche molto eterogenei, i quali non sempre possono essere facilmente modellabili analiticamente con sufficiente accuratezza. In genere non sempre tutto ciò che è simulabile di un sistema transazionale è modellabile analiticamente, se non attraverso modelli di una complessità tale da renderne la realizzazione poco vantaggiosa. In più va considerato che i modelli analitici in genere sono affidabili entro determinati intervalli di valori, e possono risultare poco precisi nei casi limite.

Quanto detto ha influito su questo lavoro sulla scelta di utilizzare, come già citato prima, un modello simulativo per validare l'affidabilità del modello analitico realizzato.

2.2.2 La caratterizzazione del workload

La caratterizzazione del carico di lavoro (*workload characterization*) è il processo atto ad identificare e modellare le caratteristiche del carico di lavoro a cui è sottoposto un sistema. Partendo da un'analisi del cosiddetto *business workload* [15] , cioè delle attività degli utenti che coinvolgono il sistema, si passa all'identificazione ed alla classificazione del *functional workload*, cioè delle richieste che gli utenti effettuano al sistema. Le transazioni determinano il *functional workload* di un sistema transazionale. L'output del processo di caratterizzazione del workload rappresenta l'input per la fase di valutazione della performance. Lavorando con un modello analitico del sistema, l'output del processo deve essere un modello analitico del *functional workload* che descrive, con un certo livello di dettaglio, il workload reale. Nel caso di sistemi transazionali un modello analitico del workload deve fornire almeno le seguenti informazioni:

- domanda di lavoro (*service demand*) delle transazioni.
- tempi di interarrivo (*interarrival time*) delle transazioni;

La domanda di lavoro rappresenta la quantità di lavoro che il sistema deve compiere per completare l'esecuzione di una transazione. Una caratterizzazione generale può essere fatta attraverso i seguenti parametri:

- lunghezza delle transazioni, misurata in numero di operazioni richieste;
- tipo di operazioni richieste (lettura, scrittura, eliminazione, inserimento);
- dataitems acceduti dalle operazioni.

Ognuno di questi parametri rappresenta un contributo al service demand delle transazioni, Il service demand di una transazione può essere ad esempio definito attraverso una tupla $<L,T,D>$, dove L rappresenta il numero di operazioni della transazione, T la sequenza ordinata dei tipi di operazioni e D la sequenza dei relativi dataitems acceduti. Quando le transazioni sono caratterizzate da valori di service demand piuttosto variabili si può effettuare una suddivisione per *classi*, $C_1,...C_n$, dove ogni C_i è caratterizzata dai propri elementi L_i, T_i, e D_i. L'approccio basato su suddivisione per classi è largamente utilizzato nella modellazione dei protocolli per il controllo di concorrenza, come in [8][9][10][11].

I tempi di interarrivo rappresentano i tempi che intercorrono tra l'arrivo della richiesta di inizio di una transazione e l'arrivo della successiva. Le variabili utilizzate nella relativa caratterizzazione dipendono innanzitutto dal modello generale di sistema. Di seguito si descrivono i due principali tipi di modelli di sistema utilizzati (*aperti* e *chiusi*) con riferimento ai sistemi transazionali e per ognuno di essi si discute della caratterizzazione dei tempi di interarrivo.

2.2.3 Modelli chiusi e modelli aperti

Nei modelli chiusi (Figura 2-1), anche detti *a popolazione finita*, vi sono un numero M utenti che generano le transazioni. Un utente, dopo il completamento di una transazione, attende un tempo detto *think time*, quindi genera la richiesta di una nuova transazione. Detto Z il valore medio del think time, l'utente genera le richieste di nuove transazioni con rate medio $1/Z$. Un utente può trovarsi in due fasi: la fase del think time, durante la quale genera le richiesta secondo quanto detto prima, e la fase in cui attende il completamento di una transazione, durante la quale non può generare nuove richieste. Dunque, il rate medio effettivo di arrivo

delle transazioni λ è dato da *(M-k)/Z*, dove k è il numero di transazioni attive nel sistema e corrisponde al numero di utenti in attesa del relativo completamento. L'inverso del rate medio, $1/\lambda$, è il tempo medio di interarrivo delle transazioni. Come si nota, entrambi dipendono dal numero di transazioni attive nel sistema. Secondo la *legge di Little* tale valore medio K si ottiene da $K=R \cdot X$, dove *R* è il response time medio delle transazioni e *X* è il throughput del sistema. Ciò comporta una dipendenza dalla performance del sistema che a sua volta dipende dai tempi di interarrivo delle transazioni. Da ciò il nome di modello chiuso.
I modelli chiusi sono adatti alla caratterizzazione di sistemi con un numero limitato di utenti, in cui il numero di utenti in attesa del completamento delle richieste effettuate può essere significativo rispetto a quello degli utenti che possono interagire con il sistema. Un esempio tipico è costituito da un sistema acceduto solo dai dipendenti di un azienda.

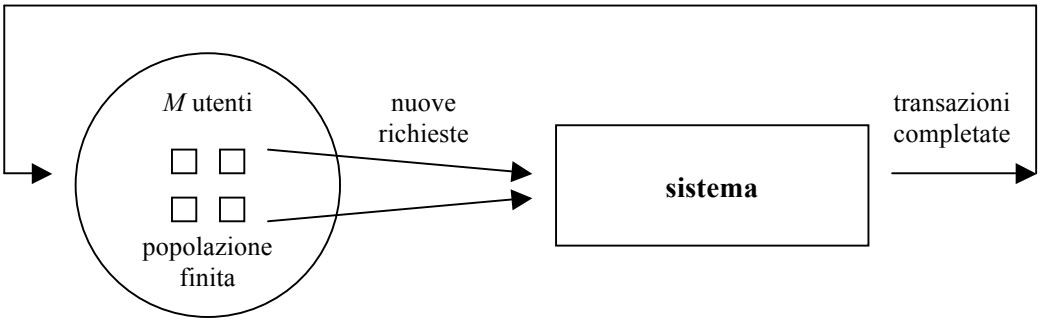


Figura 2-1. Modello chiuso

Nei modelli aperti (Figura 2-2), anche detti *a popolazione infinita*, i tempi di interarrivo delle transazioni sono indipendenti dal response time ed in generale dal throughput del sistema. In questi modelli si assume che il numero di utenti in attesa del completamento delle richieste già effettuate non influisca sull'arrivo di nuove transazioni, in quanto nuovi utenti sarebbero sempre in arrivo per effettuare nuove richieste. Dunque la caratterizzazione dei tempi di interarrivo può essere effettuata indipendentemente dalle caratteristiche del sistema. Uno strumento adatto in questo caso è rappresentato dalle distribuzioni di probabilità. Il numero di

richieste che arrivano in un determinato intervallo di tempo Δt è rappresentato da una variabile aleatoria $X(\Delta t)$. Dunque, la funzione di densità di probabilità associata alla variabile X caratterizza il processo di arrivo delle richieste.

I modelli aperti vengono utilizzati per sistemi con un elevato numero di potenziali utenti. Essi si rivelano particolarmente adatti nel caso di sistemi web.

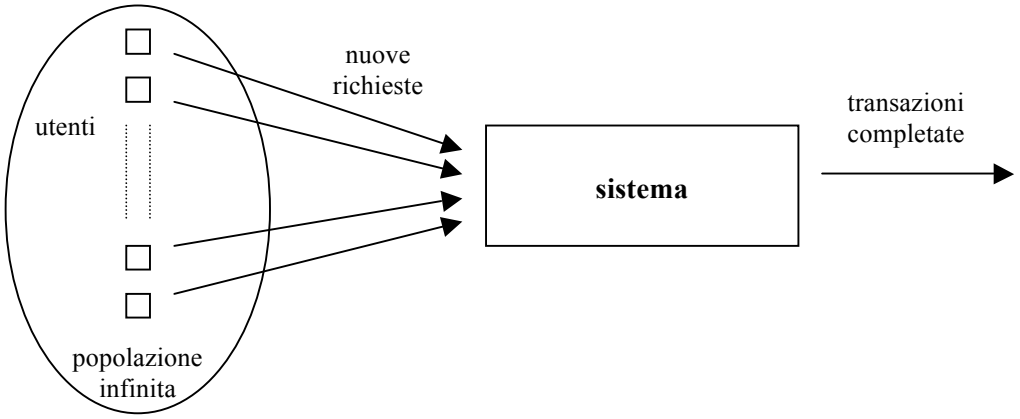


Figura 2-2. Modello aperto

2.2.4 Strumenti analitici per la costruzione di modelli

La costruzione di modelli analitici è supportata dall'utilizzo di metodi e strumenti ampiamente testati ed affidabili. Modelli più semplici possono essere realizzati attraverso metodi combinatori, basati sull'analisi e la modellazione dei singoli componenti per poi costruire un modello del sistema attraverso l'analisi delle interconnessioni tra essi. Ai fini della valutazione della performance però risultano poco adatti a causa della difficoltà nella modellazione di sistemi complessi e dinamici. A questo proposito si utilizzano modelli stocastici basati su uno spazio degli stati. In questi modelli l'evoluzione dello stato del sistema viene vista come un processo stocastico, quindi associato a distribuzioni di probabilità in funzione del tempo.

Tra i processi stocastici più adatti alla modellazione dei sistemi di elaborazione vi sono i processi di Poisson e di Markov. Il processo di Poisson è ampiamente utilizzato per modellare i processi di arrivo delle richieste nei modelli di sistema aperti per sistemi di elaborazione e di comunicazione. I processi di Markov sono utilizzati per rappresentare l'evoluzione dello stato

di un sistema, in particolare se lo spazio degli stati è di tipo discreto (catene di Markov). Questi strumenti consentono di effettuare sia analisi transienti, cioè relative ad un periodo di tempo limitato, sia stazionarie, cioè relative a periodi in cui il sistema è in equilibrio.

Alcuni casi particolari di processi di Markov, i cosiddetti processi di nascita e morte, trovano applicazione nei modelli basilari della *teoria delle code* [16]. Quest'ultima assume una fondamentale importanza ai fini della valutazione dei sistemi di elaborazione, in quanto è particolarmente adatta alla modellazione del comportamento di molti dei relativi componenti. Ad esempio un multiprocessore con n CPU, caratterizzato da un processo di arrivo delle richieste di tipo Poissoniano e distribuzione dei tempo di servizio di tipo esponenziale, può essere modellato attraverso una coda ad n serventi identificata, secondo la notazione di Kendall, come *coda M/M/n*. L'interconnessione tra componenti rappresentati da code viene modellata attraverso *reti di code.*

Infine, si citano le Reti di Petri con le relative varianti, particolarmente adatte per la modellazione delle comunicazioni e delle interazioni fra processi paralleli.

Il modello analitico realizzato in questo lavoro fa uso dei processi stocastici e della teoria delle code.

2.3 La valutazione della performance dei controlli di concorrenza

La valutazione delle prestazioni che un sistema transazionale può fornire a seconda del protocollo di concorrenza adottato è da tempo oggetto di studio della ricerca. In questo paragrafo si vogliono delinearne alcuni aspetti e sottolineare le cause che influiscono sul degrado delle prestazioni dei sistemi transazionali dovute alla presenza dei controlli di concorrenza.

Da quanto detto all'inizio del paragrafo 2.1 si può innanzitutto dedurre che l'obbiettivo del processo di valutazione delle prestazioni dei controlli di concorrenza è la valutazione della performance, non della performability. Ciò in quanto non ha senso naturalmente considerare un protocollo "soggetto a guasto". La correttezza di un protocollo per il controllo della concorrenza è generalmente verificata attraverso metodi analitici. Quando non si ha la possibilità di valutarla attraverso tali metodi si ricorre alle tecniche di *Isolation Testing* [4] [5], che consentono di rilevare le eventuali anomalie che non rispettano i livelli di isolamento

voluti. Inoltre va sottolineato che ciò che spesso è di maggior interesse, in quanto consente un confronto oggettivo rispetto ad altri protocolli, è una valutazione indipendente dalle caratteristiche del sistema in cui viene impiegato. A tal fine è necessario costruire un modello che astragga da quei fattori la cui presenza influisce sulle prestazioni indipendentemente dalle caratteristiche del protocollo stesso. D'altro canto modelli che non considerano fattori come, ad esempio, l'utilizzo della CPU, i tempi delle operazioni I/O, la probabilità di buffer hit, risulterebbero poco attendibili.

Quanto detto va tenuto conto nel processo di costruzione dei modelli analitici e nel processo di valutazione della performance attraverso tali strumenti. Questo vale soprattutto ai fini di una corretta visione dei relativi limiti e dei relativi campi di applicabilità e validità .

2.3.1 Effetti dei controlli di concorrenza sulla performance dei sistemi transazionali.

La necessità di utilizzare protocolli per il controllo della concorrenza influisce notevolmente sulla performance dei sistemi transazionali. Il tempo di risposta delle transazioni può subire incrementi rilevanti, così come il throughput del sistema può degradare considerevolmente. Ciò è dovuto principalmente alle azioni effettuate dai protocolli per garantire il livello di isolamento richiesto, ed in parte anche all'aumento del carico di elaborazione generato dall'esecuzione degli algoritmi che li implementano. Tali algoritmi devono essere attivati a seguito di molti eventi che occorrono durante l'esecuzione di ogni transazione, e possono generare notevoli carichi elaborativi e necessitare di strutture dati voluminose per la gestione dei metadati. Influiscono, quindi, sia sulla *data contention* (contesa sui dati) che sulla *resource contention* (contesa sulla risorse). I livelli di contention possono crescere sino a determinare *thrashing*, cioè il fenomeno per cui in seguito ad un aumento delle transazioni attive nel sistema non corrisponde più un incremento del throughput, quanto un degrado (Figura 2-3).

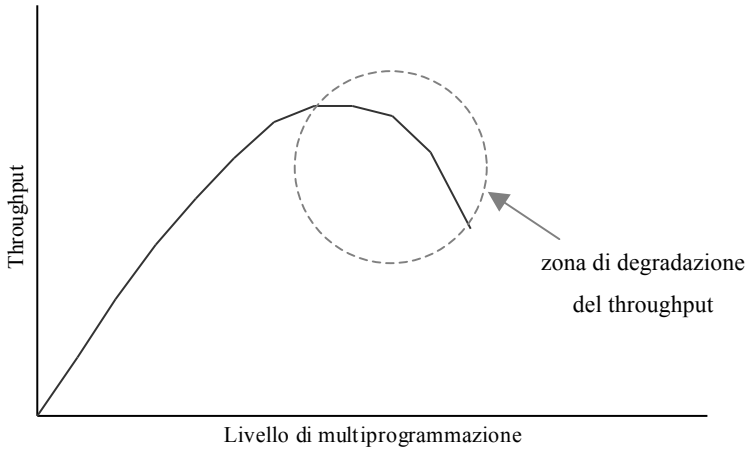


Figura 2-3. Effetti sul throughput

Si parla in particolare di *DC-thrashing* (*data contention-thrashing*) [1] quando ciò è dovuto ad alti livelli di contesa sui dati, e di *RC-thrashing* (*resource contention-thrashing*) quando è dovuto ad alti livelli di contesa sulle risorse. Per evitare fenomeni di thrashing è spesso necessario limitare preventivamente il *livello di multiprogrammazione* del sistema, ossia limitare il numero di transazione che possono essere contemporaneamente attive. Ma, ciò a sua volta, può causare attese rilevanti da parte delle transazioni in arrivo che vengono messe in coda, oltre che limitare il throughput del sistema a determinati valori. In [2] è presente un studio che attraverso simulazioni analizza intensamente la dipendenza di molte variabili del sistema rispetto dal livello di multiprogrammazione.

L'impatto sulla performance dipende particolarmente dal livello di isolamento garantito dal protocollo di concorrenza. Le prestazioni migliori corrispondono naturalmente ai livelli di isolamento più bassi, in quanto necessitano di algoritmi poco complessi. Salendo di livello occorre evitare ulteriori tipi di anomalie, dunque sono necessari algoritmi più complessi. I protocolli visti nel precedente capitolo influiscono in maniera differente sulla performance a causa delle diverse strategie adottate, in funzione dei livelli di isolamento garantiti ed a seconda delle caratteristiche del workload.

In seguito si mettono in evidenza i principali fattori responsabili del degrado della performance nel caso di strategie basate sui lock, strategie ottimistiche ed in relazione alle tecniche multiversione, come nel caso del protocollo MVCC, oggetto del lavoro di modellazione di questa tesi.

2.3.2 Performance con l'utilizzo dei locks

L'obiettivo dei protocolli di concorrenza basati sui locks è quello di prevenire l'eventuale verificarsi di anomalie. Il blocco di una transazione viene effettuato a priori, cioè senza previa verifica che l'operazione richiesta avesse potuto generare un conflitto con la transazione che manteneva il lock. Le attese fanno crescere notevolmente i tempi di risposta delle transazioni. Nella variante del two phase locking detta *strict*-two phase locking, che garantisce serializzabilità, recuperabilità ed evita il cascading rollback, i locks sono mantenuti sino al completamento dell'operazione di commit di una transazione, ed in tal modo le transazioni bloccate devono attendere il termine delle transazioni che hanno precedentemente ottenuto i lock sugli stessi dataitems.

Con l'utilizzo dei lock il fenomeno del DC-thrashing inizia a manifestarsi quando, in media, per ogni transazione attiva nel sistema ve ne è più di una in attesa [1]. In un tale scenario infatti per ogni nuova transazione in ingresso più di una ne risulterà bloccata, dunque al crescere del livello di multiprogrammazione il throughput non può che diminuire.

Molto più lievi sono gli effetti sul throughput dovuti ai deadlocks. Questi ultimi vengono risolti attraverso l'abort ed il successivo rerun delle transazioni, contribuendo ad un aumento del carico. In diversi studi, tra cui [17] e [18], si è dimostrato come la probabilità di deadlock è talmente bassa rispetto alla probabilità di contention da poter essere ignorata nel processo di valutazione della performance. Dunque gli effetti dovuti ai reruns delle transazioni causati dai deadlock possono essere considerati irrisori rispetto agli altri fattori.

Infine, un incremento dell'utilizzazione delle risorse è dovuto agli algoritmi che implementano il protocollo, quindi anche allo spazio necessario per la gestione dei locks sui dataitems e delle code di attesa delle transazioni. Questi fattori in ogni caso dipendono fortemente dalle scelte implementative relative a tali protocolli.

2.3.3 Performance con strategie ottimistiche

Le strategie di tipo ottimistico non prevengono i conflitti tra le transazioni, quanto li risolvono attraverso l'abort della transazione che ha effettuato un accesso ad un dataitem potenzialmente "dannoso". I reruns delle transazioni abortite costituiscono la principale causa della riduzione della performance. L'esecuzione dei reruns infatti, dal punto di vista delle risorse, comporta la riesecuzione di lavoro già compiuto in precedenza.

Quando il numero medio di reruns per transazione aumenta l'utilizzazione delle risorse cresce tendendo alla saturazione. A questo punto si inizia a manifestare il fenomeno dell'RC-thrashing, quindi il throughput inizia inesorabilmente a decrescere all'aumentare del numero di transazioni attive nel sistema. Quanto detto, anche in relazione all'utilizzo dei lock, trova un valido riscontro in [2].

Una strategia di tipo ottimistico può essere realizzata sia effettuando l'abort di una transazione non appena si verifica la situazione che ne è causa (protocolli di tipo *broadcast*), sia attendendo la fine della transazione (protocolli di tipo *ottimistico puro*). Con il broadcast si evita di compiere il lavoro "inutile" dovuto al completamento della transazione, mentre l'ottimistico puro trae vantaggio dal fatto che durante i reruns tutti i dataitems acceduti, con alta probabilità, saranno presenti nel buffer, a causa dell'accesso appena effettuato nel run precedente. Ciò contribuisce ad evitare attese non indifferenti dovuti ai tempi I/O, che sono di ordini di grandezza superiori rispetto alle operazioni di accesso al buffer. In [8] si mostra come questo fattore rende in alcuni casi l'ottimistico puro più performante rispetto al broadcast.

Come nel caso dei locks, l'impatto sulla performance dovuta all'utilizzazione delle risorse per l'esecuzione dell'algoritmo è legata all'implementazione degli stessi. In generale essi richiedono ulteriore memoria in particolare per la gestione dei metadati.

2.3.4 Performance con tecniche multiversione

I protocolli che utilizzano tecniche multiversione sono realizzati congiuntamente a strategie di tipo pessimistico od ottimistico. Dunque quanto già detto in relazione a queste circa l'impatto sulla performance vale anche per questi ultimi, a seconda della strategia adottata. In ogni caso le tecniche multiversione nascono con l'obiettivo di incrementare il throughput del sistema evitando di bloccare od abortire le transazioni a causa di conflitti tra letture e scritture. Ciò risulta particolarmente efficace nei casi di transazioni read-only ed in genere quando il workload è caratterizzato soprattutto da operazioni di lettura. Quest'ultima è una situazione abbastanza comune in diversi contesti, in particolare per applicazioni web. Ciò che influisce negativamente sulla performance è naturalmente il costo di gestione delle diverse versioni dei dataitems, sia in termini di tempo di accesso che di spazio. Le versioni non più utili devono essere eliminate per liberare memoria, e ciò viene fatto attraverso operazioni di verifica e

cancellazioni periodiche (*garbage collection*), che comportano ulteriori costi di elaborazione. Di contro le versioni precedenti dei dataitems possono essere sfruttate ai fini del rollback e dagli algoritmi di recovery. Per questi scopi infatti è necessario mantenere informazioni relative ai dataitems modificati, almeno per quelli acceduti dalle transazioni ancora attive. La realizzazione di un protocollo multiversione deve necessariamente tener conto di questi aspetti. I costi di gestione delle versioni e le strategie per "ammortizzarli" infatti potrebbero rivelarsi essere il punto chiave per ottenere un notevole incremento della performance.

2.3.5 Precedenti lavori e risultati

Riferendosi agli algoritmi per il controllo della concorrenza, nel 1983, Goodman, Sury e Tray in [27] riferiscono: "The correctness of these algorithms is already well understood, their performance is not".

Allo stato attuale la letteratura scientifica fornisce una discreta quantità di lavori relativi alla modellazione ed alla valutazione dei protocolli di concorrenza per i sistemi transazionali. Diversi studi riguardano strategie basate sui locks, ma anche i protocolli ottimistici sono stati spesso oggetto di analisi, nonostante i sistemi commerciali abbiano in genere mostrato preferenza per le prime, spesso utilizzandole unitamente a tecniche multiversione. Bernstein, Hadzilacos e Goodman in [1] fanno notare come, alla fine degli anni '80, non vi erano riscontri pratici sufficienti a sostegno degli studi fatti sui protocolli che non utilizzavano i locks. Successivamente, quando gli studi si sono orientati verso i confronti tra i diversi protocolli, anche i protocolli ottimistici sono stati largamente analizzati e valutati.

Gli studio dei protocolli di concorrenza presentati in letteratura sono basati fondamentalmente su modelli analitici e modelli simulativi. I modelli analitici presentati sono in genere affiancati dalle relative validazioni attraverso modelli simulativi. I modelli analitici sono stati utilizzati per valutare le prestazioni dei protocolli sia per sistemi centralizzati che distribuiti. Ad esempio Thomasian et al. propongono lavori basati su strumenti analitici in relazione a protocolli basati su locking in [9], [19] e [24], facendo riferimento a sistemi centralizzati. Ciciani, Dias e Yu, in [10] e [11], utilizzano un approccio analitico per studiare la performance di protocolli di concorrenza per database distribuiti e replicati. Yu et al., in [8], propongono una survey in cui illustrano una metodologia per la modellazione analitica basata su lavori precedenti, applicabile sia al caso di sistemi centralizzati che distribuiti. Gli approcci

basati su modelli simulativi sono stati i più adottati. In relazione ai protocolli di locking ad esempio si citano [26], [28], [29] e [30]. Molti studi effettuati riguardano confronti tra i diversi protocolli. In relazione a ciò in [2] Agrawal et al. riferiscono delle discordanze di alcuni risultati fino ad allora presentati e di come talvolta apparivano contraddittori. Quindi mostrano come ciò era fondamentalmente dovuto alle assunzioni fatte dai modelli precedenti, criticandone alcune scelte. Precedentemente Carey e Stonebraker in [20] presentano risultati ottenuti da modelli basati su reti di code per database centralizzati. Concludono sottolineando come i protocolli basati su locks mostrino performance superiori rispetto a quelli ottimistici. A conclusioni alquanto discordanti da Carey pervengono Tay in [22] e Balter et al. in [21]. Agrawal et al., nel lavoro sopra citato, mostrano attraverso modelli simulativi come i risultati siano fortemente legati alla disponibilità delle risorse di sistema. In caso di bassa disponibilità con i protocolli ottimistici il thrashing inizia a manifestarsi a livelli di multiprogrammazione più bassi, ed anche il throughput risulta inferiore. Ciò è legato all'alta utilizzazione delle poche risorse disponibili causata dai reruns delle transazioni che diventano sempre più frequenti all'aumentare della contention sui dati, fino a provocare DC-thrashing. Aumentando la disponibilità di risorse i protocolli ottimistici rivelano prestazioni migliori, fino a superare quelle dei protocolli basati su lock. Ad alta disponibilità di risorse il thrashing si manifesta difficilmente. In questo caso il protocolli pessimistici infatti sono penalizzati dal manifestarsi del DC-thrashing, che inizia a presentarsi molto prima che le risorse raggiungano alti livelli di utilizzazione. Ciò risulta pienamente concordante con quanto già detto in 2.3.2 e 2.3.3.

Allo stato attuale, in pochi dei lavori citati sui sistemi transazionali basati su lock, risulta caratterizzato in maniera specifica il workload. Questo studio si inserisce proprio in questo contesto e cerca di costruire un modello analitico per la valutazione delle performance di sistemi relazionali considerando specifici pattern di accesso ai dati delle transazioni.

Capitolo 3

Un modello analitico per il controllo di concorrenza 2PL basato su specifici pattern di accesso ai dati

In questo capitolo si presenta il modello analitico realizzato per la valutazione e la predizione della performance del 2PL (*Two Phase Locking*). Questo tipo di controllo di concorrenza è particolarmente trattato sia negli studi di letteratura scientifica che implementato nei DBMS commerciali. La letteratura fornisce alcune metodologie di analisi e diversi lavori di modellazione relativi a questo tipo di controllo, tuttavia momentaneamente non esiste nessun lavoro che focalizza l'interesse sui pattern di accesso ai dati delle classi transazioni. Gli obbiettivi di questo lavoro sono stati:

- la realizzazione del modello analitico per la descrizione del 2PL, ed in particolare lo *Strict 2PL,* con pattern di accesso ai dati non uniformi;
- la realizzazione del sistema software per il calcolo del modello analitico;
- la validazione attraverso un simulatore ad eventi discreti;

Il modello ottenuto si basa su alcune assunzioni finalizzate ad una più semplice ed efficiente modellazione del sistema.

3.1 Scenario attuale Two Phase Locking

I controlli di concorrenza basati sul Two Phase Locking (*2PL*), di cui si è discusso finora, sono quelli maggiormente usati in molti dei DBMS attualmente sul mercato. Storicamente la maggior parte dei DBMS hanno adottato la versione originale del 2PL poi successivamente raffinata con le varie estensioni tipo quella dello Strict 2PL.

DBMS come

- DB2;
- Informix;
- SQL Server;
- Sybase(ASE)

utilizzano interamente lo Strict 2PL o versioni dello stesso leggermente modificate o ottimizzate. In realtà però, grazie all'introduzione e al consolidamento di tecniche ci controllo di concorrenza più raffinate, molti degli altri DBMS commerciali che non utilizzano direttamente lo Strict 2PL adottano politiche di gestione che tendono ad unificare gli effetti positivi di più protocolli, questo finalizzato al miglioramento delle performance. Infatti se per la gestione delle scritture lo Strict 2PL è sicuramente ancora uno dei protocolli più efficace, per quanto riguarda le politiche di lettura si sono evolute altre strategie che, a seconda delle caratteristiche del workload, possono risultare più vantaggiose.

Per fare un esempio di quanto appena descritto si possono considerare alcuni dei DBMS più rilevanti sul mercato come Oracle Database che utilizza l'MVCC (*Multi Versioning Concurrency Control*), ovvero un protocollo per la gestione delle multi versioni dei dataitems in lettura ma che si poggia comunque su politiche basate su lock per la gestione delle operazioni di scrittura.

Per fare un esempio di DBMS che utilizza interamente lo Strict 2PL c'è, come detto prima, Microsoft SQL Server 2005. In realtà l'approccio pessimistico, che utilizza il 2PL come controllo di concorrenza, può essere impostato all'inizializzazione del sistema. Configurando il DBMS con questo approccio le scritture verranno implementate utilizzando lock esclusivi sui dataitems mentre le letture utilizzando dei lock condivisi. Chiaramente la presenza di un lock esclusivo non è compatibile con la presenza di un lock condiviso e viceversa. La granularità dei lock in SQL Server può essere impostata a livello di intera tabella, pagina del disco contenente i dataitems o su un blocco parziale di righe della tabella. La politica dei

deadlock implementata è quella basata sulla prevenzione dell'eventuale occorrenza di deadlock. Prima di introdurre un lock e quindi un arco nel grafo delle dipendenze delle risorse tra le transazioni, si verifica se l'eventuale lock comporti un deadlock. In tal caso la transazione viene abortita e fatta ripartire dopo un certo tempo di back-off.

3.1.1 Regole del protocollo

L'oggetto caratterizzante del Two Phase Locking è il *lock.* Con questo termine definiamo un meccanismo per evitare che se un oggetto del database è utilizzato da una transazione, nessun altra transazione può operare sullo stesso. Esistono due tipi di lock utilizzati da questo protocollo: esclusivi e condivisi (*exclusive & shared*). Grazie alla presenza dei lock condivisi, possiamo permettere letture multiple su uno stesso dataitem e bloccare le scritture sui dataitem già acceduti in lettura da altre transazioni.

Le regole del protocollo sono le seguenti:

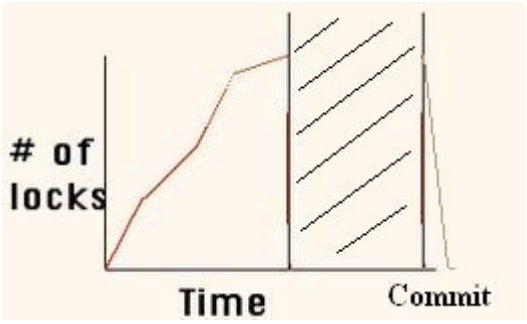
1. Quando una transazione T vuole ***scrivere*** un dataitem X viene verificata la presenza di un qualsiasi tipo di lock, esclusivo o condiviso, sul dataitem:
    1.1. In caso positivo, ovvero di lock riscontrato, si deve verificare la natura del lock:
        1.1.1. in caso di presenza di lock esclusivo posto dalla stessa transazione che vuole scrivere, si procede con la scrittura;
        1.1.2. in caso di presenza di lock esclusivo posto da un'altra transazione che vuole scrivere, quest'ultima entra in uno stato di attesa finché il lock non viene rimosso a causa di un commit o abort;
        1.1.3. in caso di presenza di lock condiviso posto dalla stessa transazione che vuole scrivere e non ci sono altre transazioni che hanno posto lock condivisi sullo stesso dataitem, si procede con la scrittura;
        1.1.4. in caso di presenza di lock condiviso posto da un'altra transazione che ha letto si entra in uno stato di attesa finché il lock non viene rimosso a causa di un commit o abort;
    1.2. in caso di dataitem non bloccato, si inserisce un lock esclusivo e si procede con la scrittura;

2. Quando una transazione T vuole ***leggere*** un dataitem X viene verificata la presenza di un qualsiasi tipo di lock, esclusivo o condiviso, sul dataitem:
    2.1. in caso positivo, ovvero di lock riscontrato, si deve verificare la natura del lock:
        2.1.1. in caso di presenza di lock esclusivo posto dalla stessa transazione che vuole leggere, quindi verosimilmente si tratta di una transazione che ha già eseguito una scrittura su X, si procede con la lettura;
        2.1.2. in caso di presenza di lock esclusivo posto da un'altra transazione, quest'ultima entra in uno stato di attesa finché il lock non viene rimosso a causa di un commit o abort;
        2.1.3. in caso di presenza di lock condiviso, si procede con la lettura;
    2.2. in caso di dataitem non bloccato, si inserisce un lock condiviso e si procede con la lettura;
3. Quando una transazione T effettua il commit rilascia tutti i locks acquisiti; se ci sono transazioni in attesa sullo stesso dataitem vengono sbloccate nell'ordine con cui hanno richiesto i locks;
4. Quando una transazione T viene abortita rilascia tutti i locks acquisiti; se ci sono transazioni in attesa sullo stesso dataitem vengono sbloccate nell'ordine con cui hanno richiesto i locks;

3.1.2 Caratteristiche Strict 2PL

Il protocollo descritto nel paragrafo precedente è il Two Phase Locking nella versione Strict 2PL ovvero Stretta. La caratteristica principale dello Strict 2PL ricade nel fatto che tutti i lock richiesti dalla transazione vengono rilasciati tutti contestualmente al commit o all'abort della stessa.

Questa particolarità rende lo Strict 2PL uno schedule capace di evitare il fenomeno del cascading Rollback. Infatti, una transazione T2 non può leggere un valore di un dataitem X scritto da T1 fin quando T1 non rilascia il lock su X, il che avviene dopo che T1 ha eseguito il commit. In figura viene mostrato un diagramma temporale che descrive il rilascio dei locks in uno schdule Strict 2PL.



Lo Strict 2PL eredita tutte le caratteristiche di Serializzabilità e di consequenza tutte le proprietà A-C-I-D del 2PL base.

3.2 Il modello analitico

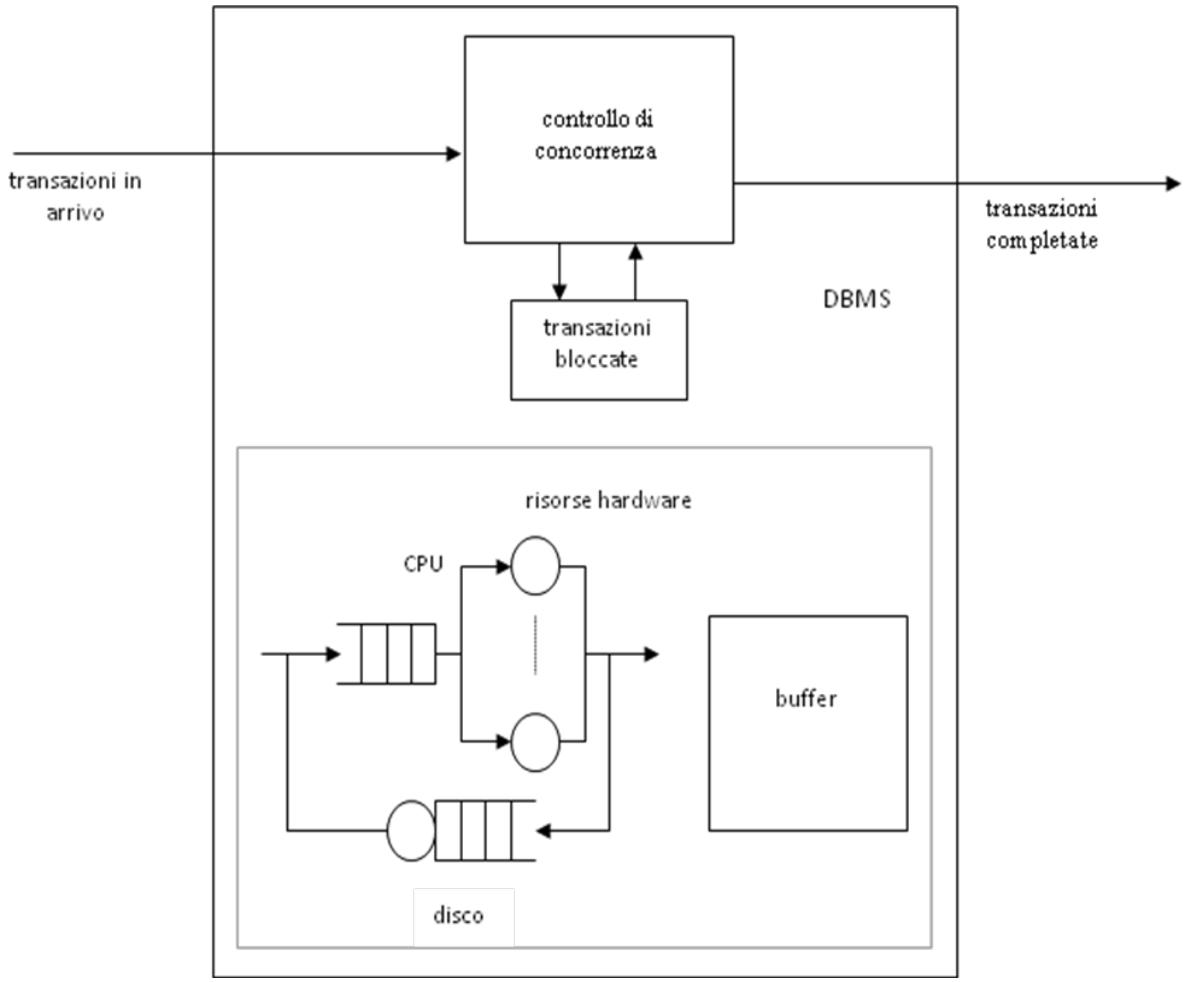
3.2.1 Modello di sistema

Il sistema in esame è costituito da un DBMS transazionale che adotta il protocollo del 2PL descritto nei paragrafi precedenti. Le transazioni possono accedere a tutti i dataitems della base di dati sia in lettura che in scrittura. La base di dati ha dimensione prefissata. Il sistema è di tipo aperto e le transazioni arrivano con un dato rate medio, caratteristico per ogni classe transazionale.

Il verificarsi di deadlock viene ignorato in quanto, come in illustrato in [10], la probabilità di ricadere in un deadlock è molto piccola rispetto la probabilità di lock contention che è fondamentale in questo tipo di modelli come mostrato successivamente. Per questo nell'analisi delle performance può essere tranquillamente ignorata.

Le risorse hardware sono costituite da k CPU di uguale potenza di calcolo, un array di m dischi e un buffer in memoria centrale di dimensioni limitate. Il processamento di una transazione prevede una serie di richieste alle risorse hardware. Ogni richiesta per una CPU è costituita da un certo numero di istruzioni macchina. Quando una transazione effettua una richiesta di lettura o di scrittura per un dataitem, se questo è già presenta nel buffer, l'operazione non necessita di accesso al disco; in caso contrario il dataitem deve essere caricato da disco tramite una richiesta di lettura. La probabilità di buffer-hit dipende, tra gli altri fattori, dalla politica di gestione dei dati del buffer manager, ed essendo tali politiche

molteplici il modello astrae da questi aspetti. A tal fine il valore medio di probabilità di buffer-hit è considerato un parametro di input per il modello.
Il modello di sistema è riportato graficamente in figura.



Modello di sistema

3.2.2 Ipotesi iniziali

Il lavoro è stato strutturato evolvendo il modello sulla base di ipotesi significative hanno avuto lo scopo di semplificare la risoluzione analitica. Successivamente alcune di queste

ipotesi sono state progressivamente rilassate per completare il modello in tutte le sue sfaccettature.

1) Esiste un'unica classe di transazioni di lunghezza fissa M;
2) La base di dati è da considerarsi suddivisa in I set di dataitems di lunghezza prefissata;
3) Gli accessi ai dataitems di un certo set sono uniformemente distribuiti su tutto il set, ovvero non ci sono dataitem più "popolari" all'interno del medesimo set;
4) Le transazioni in ogni stato effettuano soltanto un'operazione (lettura o scrittura);
5) Vengono considerate inizialmente soltanto scritture, ovvero le transazioni potranno soltanto effettuare accessi in scrittura sui dataitems richiesti.
6) E' considerato un solo livello di locking a livello di dataitem;
7) I tempi di interarrivo delle transazioni sono modellabili attraverso un processo di Poisson con rate medio di arrivo λ;
8) Il sistema è in equilibrio, ossia il rate medio di arrivo delle transazioni λ ed il throughput medio risultano uguali;
9) Ogni transazione effettua un accesso in scrittura o in lettura al più una volta su uno stesso dataitem della base di dati;
10) La probabilità di occorrenza di un deadlock è talmente bassa che nel modello non verranno considerati;
11) I set di dataitems in cui viene suddivisa la base di dati vengono considerati costituiti da un dataitem;

Queste ultime ipotesi sono considerate in diversi studi ed in particolare in relazione ai modelli analitici, tra cui in [8] e [9]. Per quanto riguarda l'ipotesi 7, come riportato in [24], e da quanto emerge da numerosi lavori, il processo di Poisson è il più utilizzato per modellare i tempi di interarrivo delle transazioni nei modelli aperti, quindi nei sistemi con largo numero di utenti. L'ipotesi 8 è giustificata dal fatto che in un sistema transazionale, se il rate medio di arrivo delle transazioni supera il throughput medio per un periodo sufficientemente lungo, il sistema è destinato a raggiungere la saturazione. Infine, in relazione all'ipotesi 9, data la scarsissima probabilità che in scenari applicativi una transazione possa effettuare due scritture sullo stesso dataitem, si ritiene trascurabile l'errore che può essere introdotto ignorando tali situazioni.

D'ora in avanti si farà riferimento alle ipotesi fatte in questo sottoparagrafo indicandone semplicemente il numero progressivo.

3.2.3 Pattern di accesso ai dati

Nel modello realizzato riveste un ruolo fondamentale la caratterizzazione del workload del sistema ed in particolare la caratterizzazione dei pattern di accesso ai dati da parte delle classi transazionali. Infatti le transazioni in un sistema vengono generalmente catalogate a seconda della sequenza con la quale accedono ai set di dataitems sulla base di dati. Come set di dataitems possono essere considerate anche, per esempio, le tabelle di un database. Con il termine pattern di accesso si intende la sequenza con la quale una classe transazionale accede ai vari set della base di dati.

Un esempio di questo è fornito nel benchmark TCP-C dove troviamo le transazioni suddivise in cinque classi transazionali. Questo tipo di suddivisione permette di caratterizzare al meglio il carico di lavoro di un sistema ed in particolare di indicare quali set (o tabelle) vengono accedute maggiormente dalle varie classi ed in che ordine. Proprio sulla base di quest'ultimo parametro evolve il modello ovvero dall'assunzione che ogni classe transazionale acceda con una certa probabilità ad un certo dataitem in un certo ordine.

Per illustrare cosa si intende per ordine con il quale una transazione accede ad un certo set di dataitem bisogna anticipare un discorso che sarà illustrato nel paragrafo successivo ovvero il fatto che ogni transazione è modellata attraverso una serie di stati. In ogni stato questa effettua una sola operazione. Dall'ipotesi 1 tutte le classi transazionali hanno lunghezza fissa quindi un numero di stati prefissato e di conseguenza, considerando l'ipotesi 4, possiamo definire l'ordine di accesso ad un dataitem come lo stato in cui la transazione ne richiede l'accesso in lettura o scrittura.

Un esempio di situazione che in cui si rende efficace un'analisi considerando i pattern di accesso è la seguente:

Considerando, semplificando in prima analisi, soltanto il caso di lock esclusivi (ipotesi 5), questo vuol dire che i dataitems su cui è stato impostato un lock nell'istante P che vengono acquisiti da una transazione di durata K rimarranno bloccati per esattamente K-P. Quindi sicuramente ci saranno alcuni dataitems, ed in particolare quelli acceduti nei primi stati delle transazioni, che rimarranno bloccati per tutta la durata di una transazione fino al commit. Se a

questa situazione aggiungiamo il fatto che ci potrebbero essere differenti classi transazionali con dei patter di accesso ai dati simili soprattutto negli stati iniziali, si capisce semplicemente come questo potrebbe comportare un degrado delle prestazioni del DBMS in termini di tempo di risposta delle transazioni assolutamente rilevante.

Dal punto di vista concreto, quanto descritto viene modellato attraverso una matrice di probabilità discrete $P_{i,k.}$ Come mostrato in figura

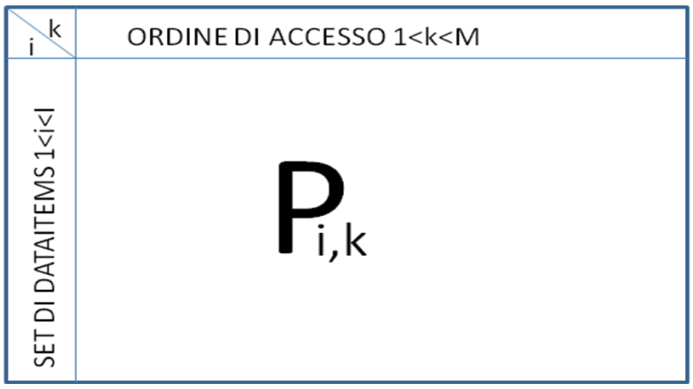


Figura 3-1. Pattern di accesso ai Set

Analizzando la matrice scopriamo che la sommatoria di ogni colonna deve essere necessariamente uguale a uno, infatti ogni transazione può accedere soltanto ad un dataitem in ogni stato. In formule:

$$\sum_{i=0}^{I} P_{i,k} \leq 1$$

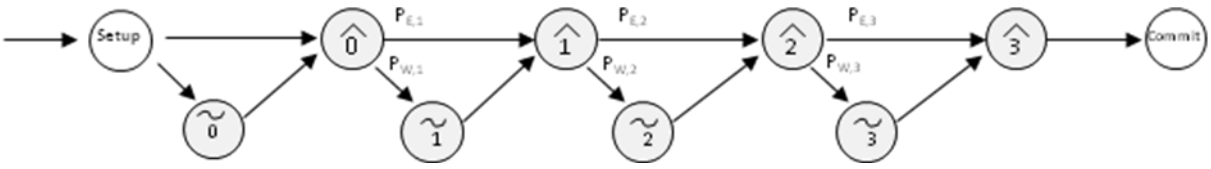
Dall'analisi delle righe di questa matrice si possono individuare i set di dataitems maggiormente acceduti dalla specifica classe transazionale.

Nella seconda parte dello studio questa matrice verrà estesa al modello con più classi transazionali, rilassando l'ipotesi 1. Per far questo verrà associata una matrice ad ogni classe transazionale presente nel sistema. In questa maniera si riusciranno a descrivere i vari pattern di accesso ai set del DB delle varie transazioni.

3.2.4 Modello di una transazione

L'esecuzione di una transazione può essere modellata attraverso un grafo orientato. I nodi corrispondono ai relativi *stati* mentre gli archi alle *transizioni* da uno stato all'altro. I nodi sono etichettati con un identificatore dello stato e gli archi con le probabilità medie di transizione. Si noti che la somma delle probabilità di transizione di tutti gli archi uscenti da un nodo deve essere uguale ad 1. Il numero di operazioni di una transazione, come detto anche in precedenza e come riportato in ipotesi 4, coincide con il numero di stati di una transazione e lo indicheremo con M. La lunghezza del grafo dipende dal numero di operazione ma in particolare per l'ipotesi 5 dal numero di scritture richieste sui dataitems.

Gli stati in cui vengono effettuate le operazioni sono contrassegnati dal simbolo '∧', mentre gli stati corrispondenti alle attese dovuti ai lock dal simbolo '~'. Il grafo completo è riportato in figura con riferimento ad una transazione con M = 3. Dal grafo si può notare come la possibilità di entrare in uno stato di wait è già possibile dallo stato di Setup. Infatti la richiesta di scrittura su un certo dataitem appartenente ad un determinato set vene fatta alla fine dello stato in cui si trova la transazione. Quindi se alla fine dello stato di setup il sistema si trova in uno dei casi: 1.1.2 oppure 1.1.4 la transazione entrerà subito in uno stato di attesa.



Grafo di una transazione con M = 3

Una transazione avanza tra gli stati del grafo nel modo seguente. Nello stato di setup effettua tutte le operazioni necessarie all'inizializzazione a seguito della richiesta di begin. Tali operazioni esulano dall'analisi del modello e verranno considerate come una serie di operazioni considerevole e accettato come parametro del modello. Nello stato $\hat{0}$ si effettua la prima operazione che per l'ipotesi 5 corrisponde ad una *richiesta* di write. A questo punto si verifica la presenza del lock sul dataitem. Per l'ipotesi 5 il lock può essere soltanto esclusivo e

per l'ipotesi 9 necessariamente il lock deve appartenere ad una transazione diversa, quindi in caso di presenza del lock prima di entrare nello stato $\hat{0}$ entriamo nello stato $\tilde{0}$ di attesa (regola 1.1.2). Se invece il dataitem non era già stato richiesto da un'altra transazione, si procede bloccando in maniera esclusiva il dataitem in questione e si esegue la scrittura (regola 1.2). La probabilità che un'operazione generi contention sullo stato 0 e che quindi porti alla situazione sopra descritta vene definita come Probabilità di wait e si indica con $P_{W,01}$ o in generale con P_W.

Se invece sul dataitem non vi è alcun lock la transazione passa nello stato $\hat{1}$ e ciò avviene con probabilità $(1 - P_w)$, che la si indica con $P_{E,1}$.

Indicheremo con P_{cont} la probabilità che nello stato $\hat{i}$ la transazione entri in uno stato di wait a seguito di un conflitto con una transazione che si trovava in uno stato $\hat{j}$. La probabilità di contention P_{cont} rivestirà un ruolo fondamentale successivamente nel modello.

Per tutti gli altri stati $\hat{i}$ con $1 \le i \le M$ si ripete lo scenario sinora descritto.

Quando la transazione entra in uno stato $\tilde{i}$ attende che il lock sul dataitem i-esimo venga rilasciato e questo avviene, per l'ipotesi 10, soltanto nel caso in cui la transazione che mantiene il lock sul dataitem effettua il commit. Allora come descritto nella regola 3 tutti i lock richiesti e concessi verranno rilasciati e le transazioni in attesa potranno sbloccarsi.

Quando una transazione raggiunge lo stato di commit effettua tutte le dovute operazioni e successivamente rilascia tutti i locks.

I tempi medi di permanenza della transazione in ogni stato $\hat{i}$ ed $\tilde{i}$ sono indicati rispettivamente con $R_{\hat{i}}$ e $R_{\tilde{i}}$. Più precisamente:

- $R_{\hat{i}}$, con $1 \le i \le M$, corrisponde al tempo di esecuzione dell'i-esima write;
- $R_{\tilde{i}}$, con $1 \le i \le M$, corrisponde al tempo di attesa di una transazione per ottenere il lock per l'i-esima scrittura.

I tempi medi di permanenza della transazione negli stati di commit e abort sono indicati rispettivamente con R_{com} e R_{abt}.

Si vuole sottolineare come i tempi $R_{\hat{i}}$, con $1 \le i \le M$, dipendano interamente dalla probabilità di buffer-hit, ossia dalla probabilità che i dataitems acceduti siano già presenti o meno sul

buffer al momento dell'accesso. In caso negativo infatti si rendono necessarie una serie di operazioni, tra cui quella di leggere il valore del dataitem dal disco. Poiché i tempi di accesso di quest'ultimo sono di ordini di grandezza superiori rispetto ai tempi del buffer, si possono verificare significativi incrementi della durata dei tempi $R_{\hat{i}}$. La probabilità di buffer-hit è legata a diversi fattori, tra cui la dimensione dello stesso e la relativa politica di sostituzione delle pagine adottata dal buffer manager. Proprio per questo si è deciso di parametrizzare questa probabilità indicandola con P_{BH}.

Probabilità di raggiungere uno stato $\hat{i}$

Come si può vedere dall'analisi del modello di transazione riportato precedentemente in figura, dal momento che una transazione effettua tutte le operazioni necessarie al setup della stessa, raggiungerà sicuramente lo stato di commit, eventualmente passando per alcuni stati di wait. Questo è possibile in relazione alle ipotesi fatte e nello specifico al fatto che l'analisi si basa su un sistema stabile in equilibrio, quindi non soggetto a errori di alcuna natura che potrebbero far abortire una transazione. Inoltre è stato escluso l'abort dovuto alla rilevazione di un deadlock. Quindi a tutti gli effetti ogni transazione inizia e sicuramente finisce. Questo ci porta ad affermare che la probabilità di raggiungere un qualsiasi stato cappello è certa. In formule:

$$\hat{\mathrm{P}}\ (\mathrm{k}) = 1, \forall k \in M$$

$$\hat{\mathrm{P}}(\mathrm{commit}) = 1$$

Response time di una transazione

Per il calcolo del response time medio di una transazione, ovvero del risultato fondamentale del modello, si utilizza il seguente approccio. Durante l'esecuzione di una transazione, essendo certa la probabilità di raggiungere gli stati cappello, verranno sommati i contributi dati dalla durata dello stato di setup iniziale della transazione, dalla durata dello stato di

commit e dalla somma del tempo di permanenza della transazione negli stati cappello e tilde. Nello specifico gli stati cappello saranno sicuramente eseguiti tutti, quindi a questi verranno sommati gli stati tilde (di attesa) pesati per la relativa probabilità di essere raggiunti, ovvero P_w. In formule avremo:

$$R = R_{setup} + \hat{R}_0 + \sum_{k=1}^{M} \left(\hat{R} + \tilde{R}_k^0 \right) + R_{commit} \quad (1)$$

Che semplificata considerando costante $\hat{R}$ sarà:

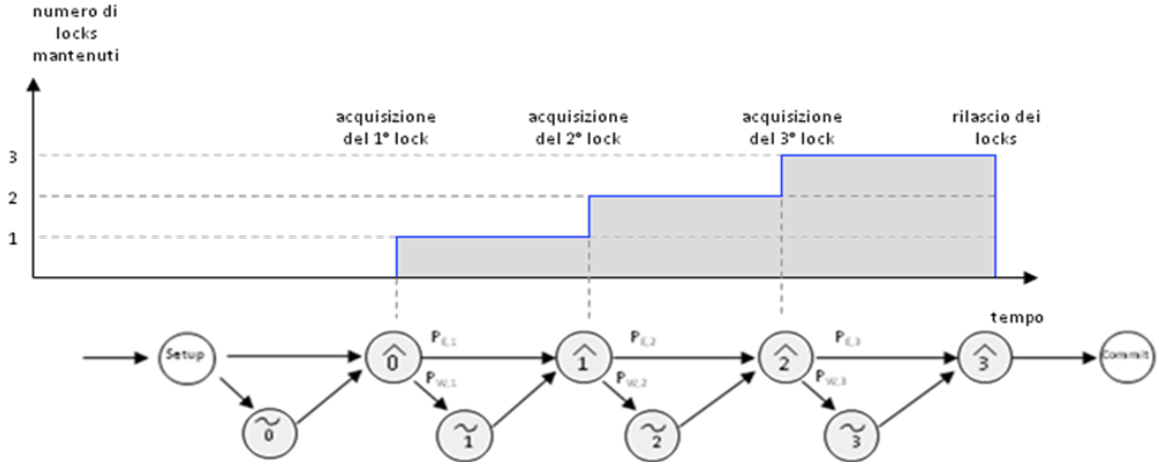
$$R = R_{setup} + M * \hat{R} + \sum_{k=1}^{M} \left(\tilde{R}_k^0 \right) + R_{commit} \quad (2)$$

Dove in $\tilde{R}_k^0$ sarà considerata non soltanto la probabilità di raggiungere lo stato k tilde ma anche la probabilità, descritta nei paragrafi precedenti, di toccare uno specifico set di dataitem nello stato k. Sull'espressione di $\tilde{R}_k^0$ si tornerà nei paragrafi successivi.

Utilizzazione dei set di dataitems

Una transazione acquisisce un lock sull'i-esimo dataitem all'interno di un set all'inizio di uno stato $\hat{i}$ e lo mantiene sino al termine della transazione. Quando raggiunge lo stato di commit allora rilascia tutti i locks. Inoltre si assume che tutti i lock siano rilasciati contemporaneamente alla fine. Nell'ambito del modello adesso siamo interessati a capire quanto, mediamente, ogni set di dataitems rimane bloccato essendo stato posto un lock su di esso. Chiaramente questo tempo dipenderà fortemente dall'istante in cui la transazione ne ha richiesto il lock, ovvero dallo stato in cui si trovava al momento della richiesta. Infatti se una classe transazionale lunga M stati richiede un'operazione su un dataitem nello stato k, visto che il modello implementa lo Strict 2PL, questo dataitem rimarrà bloccato fino al raggiungimento dello stato di commit quindi per M-k stati.
Considerando quanto detto poco sopra, se i dataitems vengono richiesti (e quindi bloccati) all'inizio delle transazioni, la durata dei locks su questi sarà sicuramente elevata. Questo influenza molto l'utilizzazione.

Questa dipendenza stretta tra il set di dataitems considerato e l'istante in cui ne viene richiesto l'utilizzo verrà modellata utilizzando la probabilità $P_{i,k.}$ introdotta sopra. La figura di seguito aiuta a capire come l'istante di richiesta del lock influisca sull'utilizzazione totale del set.



Nella pratica avremo un valore Th_i per ogni set di dataitems, calcolato sommando la durata del lock considerando che la richiesta avvenga nei vari stati, pesata per la probabilità di accedere allo specifico set nei vari istanti considerati. In formule avremo:

$$Th_i = \sum_{k=0}^{M} \left[P_{i,k} * \left(\sum_{j=k}^{M} \hat{R} + \sum_{j=k+1}^{M} R_j \right) \right] + R_{commit} * \sum_{k=0}^{M} P_{i,k} \qquad (3)$$

Che semplificando seguendo le indicazioni già date avremo:

$$Th_i = \sum_{k=0}^{M} \left[P_{i,k} * \left((M-k) * \hat{R} + \sum_{j=k+1}^{M} R_j \right) \right] + R_{commit} * \sum_{k=0}^{M} P_{i,k} \qquad (4)$$

Questo valore è il tempo medio che ogni set di dataitems trascorrerà bloccato da un lock posto da una qualche transazione che ha richiesto una scrittura su di esso. Quindi avremo un insieme di valori di Th_i.

3.2.5 Modello della contesa sui dati

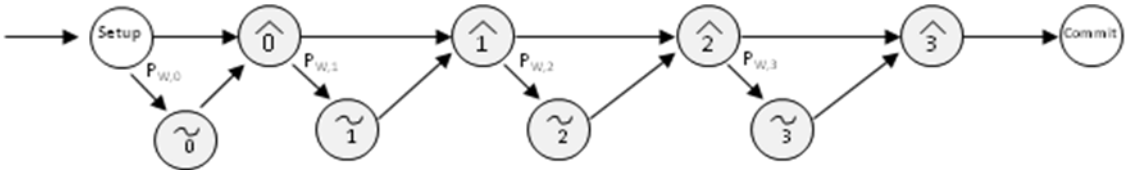
In questa parte del modello viene valutato l'impatto della data contention sul tempo totale di esecuzione della transazione. Per data contention si intende l'evento che si presenta nel momento in cui una transazione richiede un'operazione su un certo dataitem che è già stato acceduto ad un'altra transazione. Il risultato della data contention è far entrare la transazione che ha richiesto l'operazione sul dataitem in un stato di attesa che abbiamo già chiamato $\tilde{i}$.
In generale una modellazione accurata della data contention rende estremamente realistico il modello e maggiormente applicabile a situazioni di carico reali, infatti le performance di un sistema transazionale sono fortemente influenzate dalla data contention delle transazioni.

Probabilità di wait

Il primo parametro da considerare per l'analisi della data contention è la probabilità di wait, indicata con P_{wi}. Questo valore esprime con quale probabilità una transazione che richiede un'operazione sul dataitem i, rischia di entrare in uno stato di wait a causa di un lock già esistente su i. Questa probabilità, una per ogni set di dataitems, coincide con l'utilizzazione del singolo set di dataitems. Infatti il fatto che un set di dataitems abbia una alta probabilità di wait vuol dire che è fortemente richiesto dalle transazione e quindi maggiormente utilizzato.
In termini pratici questo indice di utilizzazione viene calcolato moltiplicando il tempo di utilizzo del singolo set di dataitems con la frequenza con la quale le transazioni accedono a quel set. Seguendo l'ipotesi 1 esiste una sola frequenza caratteristica dell'unica classe transazionale esistente che è rappresentata da λ. In formule avremo:

$$P_{Wi} = \lambda * \mathrm{Th}_{\mathrm{i}} \qquad (5)$$

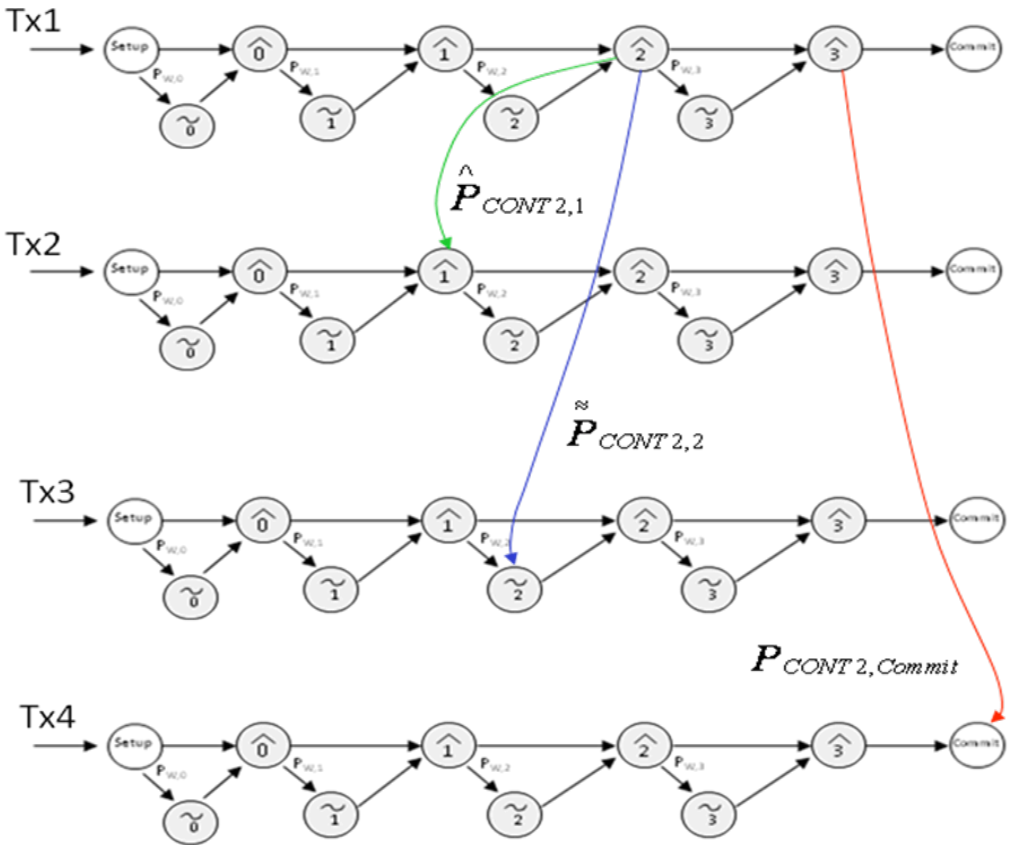
P_{Wi} verrà utilizzata in seguito nel calcolo del tempo medio di durata di uno stato tilde, infatti ricordando il modello di transazione adottato possiamo individuare la probabilità di wait P_{wi} negli archi uscenti dagli stati cappello ed entranti negli stati tilde. Ovviamente la probabilità calcolata precedentemente è relativa allo specifico set di dataitems i e per capire maggiormente il significato, si assume nel grafo sottostante che un'operazione richiesta nello stato i-cappello richieda un lock sul set i-esimo di dataitems.



Probabilità di contention

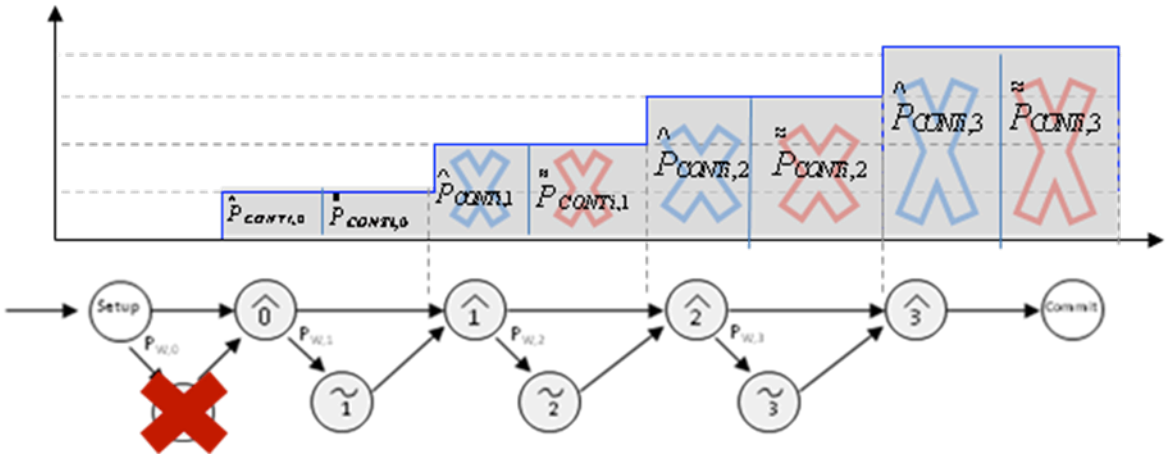
La probabilità di contention è molto differente dalla probabilità di wait appena studiata. Con probabilità di contention si intende la probabilità che, richiesto un lock sul set i-esimo, la transazione richiedente può andare in conflitto con un'altra transazione in esecuzione nel sistema che è già in possesso del lock richiesto e che nel momento della richiesta si trova nello stato di esecuzione k. Per illustrare nel dettaglio questa probabilità si consideri un esempio di transazione tx1 lunga 10 stati che accede al set di dataitems i-esimo nello stato i. Un'altra transazione tx2 successivamente entra nel sistema e richiede nel suo stato 0 un dataitem del set 0. Al momento della richiesta la transazione tx1 si trova nello stato $\hat{3}$ quindi ancora non ha effettuato il commit e rilasciato tutti i lock di cui è in possesso. In queste condizioni tx2 entrerà nello stato $\tilde{0}$ fino al termine di tx1. Questa condizione è rappresentata nel modello analitico come la probabilità che una transazione (nell'esempio tx2) a seguito della richiesta di un dataitem nel set i (nell'esempio 0) entri in conflitto con un'altra transazione (nell'esempio tx1) che si trova nello stato di esecuzione k (nell'esempio 3). Chiameremo questa probabilità $P_{CONTi,k}$.

Ripercorrendo la definizione di P_{CONT} ci si accorge che in realtà questa probabilità può essere maggiormente specializzata. Infatti si potrebbe essere più precisi nell'indicare la tipologia dello stato in cui si trova la transazione con cui si è entrati in conflitto. Dalla figura vengono mostrate le tre probabilità di contention corrispondenti ai tre possibili stati in cui si può trovare la transazione che ci blocca: cappello, tilde o commit.



Da questa figura e da quanto detto possiamo osservare come, in presenza di set di dataitems particolarmente acceduti negli stati iniziali delle varie transazioni, il sistema sarà costretto ad attendere fino al commit e quindi al rilascio, degradando il throughput e quindi i tempi di risposta delle singole transazioni.

Quanto illustrato dal punto di vista teorico, verrà praticamente calcolato attraverso un particolare ragionamento matematico di seguito illustrato:



Come prima cosa osserviamo che lo stato 0 tilde è stato escluso dal ragionamento, infatti è impossibile entrare in conflitto con una transazione che è nello stato zero tilde in quanto questa non può ancora aver bloccato nessun dataitem in quanto non ha eseguito ancora nessuna operazione.

Prendiamo in esame $\hat{P}_{CONTi,k}$. Questa probabilità può essere calcolata facendo un rapporto tra la somma delle aree occupate dagli stati cappello fino allo stato k, diviso la somma di tutta l'area sottesa dalla curva a gradino in figura. Quest'area rappresenta il tempo totale di esecuzione di tutti gli stati pesato per la probabilità di toccare lo specifico dataitem i nei vari stati. Esprimendo in formle il denominatore di questo rapporto, ovvero l'area totale sottesa, si avrà:

$$\sum_{k=0}^{M-1}\left[\sum_{l=0}^{K} P_{i,l} * \hat{R}\right] + \sum_{k=0}^{M-1}\left[\sum_{l=0}^{K} P_{i,l} * \tilde{\approx}{R}_{l}\right] + \left[\sum_{k=0}^{M} P_{i,l} * R_{commit}\right] \quad (6)$$

. Completando il calcolo di $\hat{P}_{CONTi,k}$ si avrà di conseguenza:

$$\hat{P}_{CONTi,k} = \frac{\sum_{l=0}^{K} P_{i,l} * \hat{R}}{\sum_{k=0}^{M-1}\left[\sum_{l=0}^{K} P_{i,l} * \hat{R}\right] + \sum_{k=0}^{M-1}\left[\sum_{l=0}^{K} P_{i,l} * \tilde{\approx}{R}_{l}\right] + \left[\sum_{k=0}^{M} P_{i,l} * R_{commit}\right]} \quad (7)$$

Analogamente anche $\tilde{\approx}{P}_{CONTi,k}$ può essere calcolata facendo un rapporto tra la somma delle aree occupate dagli stati tilde fino allo stato k, diviso la somma di tutta l'area sottesa dalla curva a gradino in figura. Il denominatore è rappresentato quindi dallo stesso riportato nell'espressione precedente quindi la formula generale sarà:

$$\tilde{\tilde{P}}_{CONTi,k} = \frac{\sum_{l=0}^{K} P_{i,l} * \tilde{\tilde{R}}_k}{\sum_{k=0}^{M-1}\left[\sum_{l=0}^{K} P_{i,l} * \hat{R}\right] + \sum_{k=0}^{M-1}\left[\sum_{l=0}^{K} P_{i,l} * \tilde{\tilde{R}}_l\right] + \left[\sum_{k=0}^{M} P_{i,l} * R_{commit}\right]} \quad (8)$$

Per il calcolo di $P_{CONTi,Commit}$ il concetto rimane lo stesso utilizzato per le altre due espressioni con la variante che adesso non si avrà più la dipendenza da K. Infatti $P_{CONTi,Commit}$ può essere calcolata facendo un rapporto tra la somma delle aree occupate dagli stati cappello fino allo stato di commit, diviso la somma di tutta l'area sottesa dalla curva a gradino in figura. Il denominatore sarà nuovamente rappresentato dallo stesso riportato prima e quindi l'espressione generale sarà:

$$P_{CONTi,Commit} = \frac{\sum_{l=0}^{M} P_{i,l} * R_{commit}}{\sum_{k=0}^{M-1}\left[\sum_{l=0}^{K} P_{i,l} * \hat{R}\right] + \sum_{k=0}^{M-1}\left[\sum_{l=0}^{K} P_{i,l} * \tilde{\tilde{R}}_l\right] + \left[\sum_{k=0}^{M} P_{i,l} * R_{commit}\right]} \quad (9)$$

Tempi di attesa su i set di dataitems

Quando una transazione tenta un accesso ad un dataitem su cui è già attivo un lock entra in uno stato di attesa. Come descritto anche in precedenza lo stato $\tilde{i}$ rappresenta proprio questo stato. In questo paragrafo ci occuperemo di trovare una maniera per esprimere il tempo medio di attesa su ogni set di dataitems. Utilizzando le probabilità di contention calcolate precedentemente, arriveremo all'espressione generale di $\tilde{\approx}_{Wi}$ ovvero, continuando ad utilizzare la convenzione per cui tutti i tempi vengono etichettati con R, al tempo medio che una transazione dovrà spendere in uno stato di wait se richiede lo specifico dataitems i. Questo tempo sarà pesato sia sulle probabilità di contention che sugli stati in cui il dataitem viene richiesto. La formula sarà pertanto costituita da tre blocchi, uno per ogni tipologia di stato. Avremo quindi:

$$\begin{aligned}\tilde{\approx}_{Wi} &= \sum_{k=0}^{M} \hat{P}_{CONTi,k} * \left(\frac{\hat{R}}{2} + \sum_{l=k+1}^{M} \hat{R} + \sum_{l=k+1}^{M} \tilde{\approx}_{l} + R_{commit} \right) + \\ &+ \sum_{k=1}^{M} \tilde{\approx}_{CONTi,k} * \left(\frac{\tilde{\approx}_{k}}{2} + \sum_{l=k+1}^{M} \hat{R} + \sum_{l=k+1}^{M} \tilde{\approx}_{l} + R_{commit} \right) + \\ &+ P_{CONTi,Commit} * \frac{\hat{R}}{2}\end{aligned} \tag{10}$$

Che può essere semplificata così:

$$\tilde{\tilde{R}}_{Wi} = \sum_{k=0}^{M} \hat{P}_{CONTi,k} * \left(\frac{\hat{R}}{2} + (M - l - 1) * \hat{R} + \sum_{l=k+1}^{M} \tilde{\tilde{R}}_l + R_{commit} \right) +$$

$$+ \sum_{k=1}^{M} \tilde{\tilde{P}}_{CONTi,k} * \left(\frac{\tilde{\tilde{R}}_k}{2} + (M - l - 1) * \hat{R} + \sum_{l=k+1}^{M} \tilde{\tilde{R}}_l + R_{commit} \right) + \quad (11)$$

$$+ P_{CONTi,Commit} * \frac{\hat{R}}{2}$$

Dato che tutti gli $\hat{R}$ sono indipendenti dallo stato k e dipendono soltanto dal modello hardware.

Tempi di attesa sugli stati tilde

L'espressione di $\tilde{\tilde{R}}_{Wi}$, come detto anche prima, è legata al set di dataitems considerato mentre adesso cerchiamo un'espressione del tempo di attesa relativa soltanto allo stato ed indipendente dal set considerato. Questo perché nell'espressione generale riportata nei primi paragrafi, per calcolare il tempo di risposta medio di una transazione si sommano su tutti gli stati della transazione, i contributi dati dagli stati cappello e tilde, sommando alla fine anche lo stato di commit. Quindi l'espressione di $\tilde{\tilde{R}}_{Wi}$ dipendente dal set di dataitems considerato dovrà essere normalizzata su tutti i dataitems e pesata per la probabilità di toccare il set i-esimo nello stato k-esimo.

Indichiamo questo valore con $\tilde{\tilde{R}}_k$ ovvero il tempo di attesa medio speso da una transazione nello stato $\tilde{\tilde{k}}$, quindi indipendente dal set di dataitems toccato. L'espressione che descrive questo tempo sarà:

$$\tilde{\tilde{R}}_k = \sum_{i=0}^{I} P_{i,k} * \tilde{\tilde{R}}_{Wi} * P_{Wi} \qquad (12)$$

Nell'espressione possiamo notare come per ogni set di dataitems moltiplichiamo il tempo di attesa $\tilde{\tilde{R}}_{Wi}$ che la transazione eventualmente trascorrerà nello stato di wait per accedere al set i-esimo, con l'effettiva probabilità di entrare nello stato di wait a seguito di una richiesta sul set i. Questo prodotto verrà poi pesato con la probabilità che ha la transazione di toccare il set i-esimo nello stato k.

3.2.6 Modello delle risorse hardware

Le risorse hardware sono costituite da:

- k CPU;
- un array di m dischi;
- un buffer in memoria centrale di dimensioni limitate;

Ad ogni stato di una transazione corrispondono una serie di richieste verso una CPU ed eventualmente un disco. Le CPU sono modellate attraverso una coda M/M/k. La base di dati si assume distribuita tra m dischi, dunque ogni operazione che necessita di un accesso al disco determinerà una richiesta per uno solo di essi. La dimensione del buffer risulta in ogni caso inferiore alla dimensione della base di dati, dunque la probabilità media di buffer-hit risulterà in genere inferiore ad 1. Si ribadisce che il modello non entra in merito alla politica di gestione delle pagine sul buffer. A tal fine prende in input il valore di probabilità di buffer-hit P_{BH} descritto in precedenza.

Le interazioni tra le CPU ed i dischi possono essere modellate attraverso una rete di code. La valutazione di modelli di questo tipo è largamente trattata nella teoria delle code. Per questo motivo ci si limita a mostrare come è possibile utilizzare un modello in cui si considera una coda M/M/k per le CPU ed un ritardo deterministico per un accesso ad un disco.

Si definiscono i seguenti valori:

- I_{Setup} numero medio di istruzioni macchina per il setup di una transazione;
- I_{Write} numero medio di istruzioni macchina per un'operazione di write;
- I_{Read} numero medio di istruzioni macchina per un'operazione di read;
- P^R probabilità di incontrare un'operazione di read nell'esecuzione della transazione;
- P^W probabilità di incontrare un'operazione di write nell'esecuzione della transazione;
- I_{Commit} numero medio di istruzioni macchina per il commit di una transazione;

Il carico di CPU dovuto all'esecuzione di una transazione è

$$C = I_{Setup} + \sum_{k=0}^{M} \left[\left(P^R * I_{READ} \right) + \left(P^W * I_{WRITE} \right) \right] + I_{Commit} \qquad (13)$$

Come si nota le istruzioni relative alle read e alle write sono pesate per la probabilità di incontrare rispettivamente una read o una write durante tutte le M operazioni eseguite da una transazione.

L'utilizzazione delle CPU è quindi

$$\rho = \frac{\lambda \cdot C}{k \cdot MIPS} \qquad (14)$$

Dalla teoria delle code si ha ([16]):

$$p[attesa] = \frac{\left(\frac{(k\rho)^K}{k!} \cdot \frac{1}{1-\rho} \right)}{\left[\sum_{j=0}^{k-1} \frac{(k\rho)^j}{j!} + \frac{(k\rho)^K}{k!} \cdot \frac{1}{1-\rho} \right]} \qquad (15)$$

E sempre di conseguenza:

$$\gamma = 1 + \frac{\mathrm{p[attesa]}}{\mathrm{k} \cdot (1 - \rho)} \tag{16}$$

Sulla base dei valori appena calcolati possiamo rappresentare il tempo medio di esecuzione degli stati di setup e di commit rispettivamente primo e ultimo stato di esecuzione di una transazione. Questi saranno:

$$\mathrm{R}_{Setup} = \gamma \cdot \frac{\mathrm{I}_{Setup}}{\mathrm{MIPS}} \tag{17}$$

$$\mathrm{R}_{Commit} = \gamma \cdot \frac{\mathrm{I}_{Commit}}{\mathrm{MIPS}} \tag{18}$$

Per quanto riguarda il tempo medio di esecuzione degli stati cappello, questo verrà influenzato in maniera preponderante dal tempo di accesso al disco. Infatti negli stati cappello la transazione esegue le operazioni di write, quindi dovrà interagire con il disco. In questo ambito si inserisce la probabilità di hit sul buffer P_{BH}. In funzione di questo il response time degli stati cappello può essere formalizzato come segue:

$$\hat{R} = \gamma \cdot \frac{\left(P^R * I_{READ}\right) + \left(P^W * I_{WRITE}\right)}{\mathrm{MIPS}} + \left[T_{IO} * \left(1 - P_{BH}\right)\right] \tag{19}$$

Una considerazione importate da fare ricade nel fatto che $\hat{R}$ è indipendente da tutti i parametri sopra calcolati. L'unica ovvia dipendenza è dal. modello hardware che si utilizza alla base del sistema transazionale. Questa osservazione viene utilizzata durante le semplificazioni nelle formule (1), (3), (11) trasformando:

$$\sum_{l=1}^{M} \hat{R} \quad \text{in} \quad (M-1)*\hat{R}.$$

3.2.7 Risoluzione numerica del modello

Un possibile approccio alla risoluzione numerica del modello analitico è descritto in seguito.

Parametri di input

I valori di input sono i seguenti:

I	numero di set di dataitems con cui è divisa la base di dati;
λ	rate medio di arrivo delle transazioni;
M	lunghezza di una transazione;
I_{Setup}	numero medio di istruzioni macchina per il setup di una transazione;
I_{Write}	numero medio di istruzioni macchina per un'operazione di write;
I_{Read}	numero medio di istruzioni macchina per un'operazione di read;
I_{Commit}	numero medio di istruzioni macchina per il commit di una transazione;
N_{CPU}	numero di CPU disponibili;
MIPS	milioni di operazioni al secondo eseguite da una CPU;
T_{IO}	tempo medio per una istruzione di IO su disco;
P_{BH}	probabilità media di buffer-hit;

Inzializzazione

Si inizia considerando il sistema scarico, dunque tutte e tre le P_{cont} si pongono uguali a 0.

$$\hat{P}_{CONTi,k} = 0 \qquad \forall i \in I, \forall k \in M$$

$$\tilde{\tilde{P}}_{CONTi,k} = 0 \qquad \forall i \in I, \forall k \in M$$

$$P_{CONTi,Commit} = 0 \qquad \forall i \in I$$

Inoltre essendo lo stato iniziale si pongono a 0 tutti gli R tilde, quindi

$$\tilde{\tilde{R}}_{Wi} = 0 \qquad \forall i \in I$$

E di conseguenza considerando la (12) avremo:

$$\tilde{\tilde{R}}_{k} = 0 \qquad \forall k \in M$$

Procedimento iterativo

Prima di procedere con il procedimento iterativo vengono calcolati quei valori che rimarranno costanti nelle varie iterazioni ovvero il modello hardware, $\hat{R}$,R_{Setup} e R_{Commit}. Per far questo rispettivamente verranno usate le formule:

- Carico per transazione con la (13);
- Utilizzazione con la (14);
- p[Attesa] con la (15);
- γ con la (16);
- R_{Setup} con la (17);
- R_{Commit} con la (18);
- $\hat{R}$ con la (19);

Per il procedimento iterativo si procede come segue:

Attraverso la (4) si calcola l'utilizzazione per ogni set di dataitems. Questa dipende da $\tilde{\tilde{R}}_{k}$ che nella prima iterazione è uguale a 0, come descritto in inizializzazione. Successivamente con la (5) viene calcolata la probabilità di wait P_{Wi} su ogni singolo set di dataitems. Quindi entrambe le (4) e (5) verranno applicate per ogni 0 < i < I.

Sulla base di questi parametri si calcolano poi le tre probabilità di contention attraverso la (7) per $\hat{P}_{CONTi,k}$, la (8) per $\tilde{\tilde{P}}_{CONTi,k}$ e la (9) per $P_{CONTi,Commit}$. Queste tre probabilità dovranno essere calcolate per ogni 0 < k < M e per ogni 0 < i < I.

All'interno di queste probabilità è presente la dipendenza con gli $\tilde{\tilde{R}}_k$ calcolati nell'iterazione precedente.

Adesso possono essere calcolati i tempi di attesa su tutti i set dataitems attraverso la (11). Questa formula è un punto di raccordo tra alcune delle formule appena calcolate, infatti $\tilde{\tilde{R}}_{Wi}$ dipende sia dalle tre Probabilità di contention che dagli $\tilde{\tilde{R}}_k$.

Dopo aver calcolato $\tilde{\tilde{R}}_{Wi}$ per ogni 0 < i < I si può calcolare il vettore di $\tilde{\tilde{R}}_k$, uno per ogni stato delle transazioni utilizzando la (12). Questa formula riveste un ruolo fondamentale non soltanto perché riunisce tutte quelle sopra calcola tema anche perché è il punto di passaggio tra le varie iterazioni. Infatti tutte le formule descritte nel processo iterativo dipendono dal valore di $\tilde{\tilde{R}}_k$ calcolato nell'iterazione precedente.

Con tutti i parametri possiamo finalmente calcolare il response time medio per ogni iterazione utilizzando la (2).

Al termine del procedimento descritto occorre inserire il test per la terminazione delle iterazioni realizzato come segue. Si stabilisce l'errore massimo desiderato in relazione alle variabili per cui si è interessati. Alla fine di ogni iterazione si confronta il valore della variabile con quello che era all'inizio dell'iterazione. Se la differenza è inferiore rispetto all'errore stabilito il calcolo termina altrimenti prosegue con un'altra iterazione. Nel calcolo numerico del modello è stato utilizzato come parametro di confronto il response time di una transazione. Considerando ε come l'errore massimo accettato, R^{i-1} è il valore del response time all'inizio dell'i-esima iterazione ed R^{i} è lo stesso valore al termine della stessa, allora il predicato di terminazione sarà $\left(R^{i}-R^{i-1} < \varepsilon\right)$. Se il predicato risulta falso si riapplica il procedimento utilizzando i nuovi valori calcolati per tutte le variabili, altrimenti si è raggiunta

la convergenza e si terminano le iterazioni. Nel caso del calcolo del modello appena descritto la convergenza si raggiunge in 5/6 iterazioni.

3.2.8 Modello esteso

Si vuole estendere ora il modello appena descritto nel senso che verranno rilassate alcune delle ipotesi maggiormente significative che hanno inciso molto sulla semplificazione del modello base. In questo senso l'ipotesi 1 è sicuramente quella che ha inciso maggiormente e che andremo ora a rilassare. Il suo rilassamento comporterà una seria complicazione del modello per questo si è deciso di affrontare prima il modello base escludendo questa ipotesi e successivamente approfondirlo rilassandola. Per quanto riguarda le altre ipotesi elencate, possiamo considerare la 2, la 4, la 7, la 8, la 9 e la 10 come ipotesi assolutamente non limitative che non influiscono pesantemente nell'accuratezza del modello bensì considerarle sarebbe probabilmente un'inutile complicazione. Altra ipotesi importante che si cercherà di rilassare è la 5 ovvero la presenza nel sistema soltanto di operazioni di scrittura sui dataitems.

Classi transazionali

Ricordando lo scopo principale di questo lavoro, ovvero la modellazione accurata dei pattern di accesso ai dati da parte delle transazioni, il modello base forniva le primitive per descrivere questo comportamento ma la mancanza di più classi transazionali portava una certa limitazione nei test di validità. Rilassando il vincolo della singola classe transazionale adesso andremo a modificare il modello con la finalità di caratterizzare in maniera migliore il pattern di accesso ai dati.

Prima di tutto l'introduzione delle classi transazionali comporta una serie di migliorie che rendono il modello molto accurato e maggiormente applicabile. Infatti ogni classe transazionale avrà una lunghezza propria indipendente dalle altre classi e potrà rispettare specifici patter di acceso alla base di dati, al contrario del modello base in cui la classe transazionale era unica di lunghezza prefissata Un esempio concreto di questo tipo di classificazione delle transazioni si trova nel benchmark TPC-C [23] in cui vengono definite 5 classi transazionali caratterizzando ognuna sulla base della semantica della transazione, ovvero sul significato, della lunghezza e della sequenza di accesso ai dati. Ricollegandomi proprio a quanto appena detto, la caratterizzazione dei pattern di accesso permette di mettere in risalto la sequenza con la quale ogni classe transazionale richiede un'operazione su un

dataitem. L'ordine rappresenta l'elemento chiave di questa estensione in quanto in presenza di set di dataitems acceduti da molte classi transazionali nello stesso ordine il sistema subisce un degrado delle prestazioni notevole con il verificarsi di colli di bottiglia pericolosi e difficili da gestire.

In funzione di tutto questo caratterizzeremo ogni singola classe transazionale con alcuni parametri che prima erano unici e che adesso diventano un insieme ordinato di valori. Questi sono:

- Frequenza di arrivo delle transazioni di una singola classe;
- Probabilità che una specifica classe transazionale acceda ad un certo set i di dataitems nell'ordine k;
- Lunghezza specifica di ogni classe transazionale;

Attraverso la seconda verranno modellati i pattern di accesso delle singole classi.

Quindi ricapitolando si avranno i seguenti parametri modificati rispetto al modello base:

- T Numero di classi transazionali;
- λ^c frequenza di arrivo delle transazioni della classe c;
- M^c lunghezza delle transazioni della classe c, ovvero numero di operazioni per le tx che appartengono a c;
- $P_{i,k}^c$ probabilità che una transazione di classe c acceda al set di dataitems i nell'ordine k;
- R^c il response time medio della classe transazionale c;
- Th_i^c l'utilizzazione media del set i da parte della classe c;
- $\hat{P}_{CONTi,k}^{c}$ probabilità che, dato la richiesta del lock sul dataitem i-esimo, di entrare in conflitto con una transazione di classe c nello stato k cappello;
- $\tilde{P}_{CONTi,k}^{c}$ probabilità che, dato la richiesta del lock sul dataitem i-esimo, di entrare in conflitto con una transazione di classe c in attesa nello stato k tilde;

- $P^{c}_{CONTi,Commit}$ probabilità che, dato la richiesta del lock sul dataitem i-esimo, di entrare in conflitto con una transazione di classe c in attesa nello stato di commit;
- $\tilde{R}^{c}_{k}$ tempo di attesa medio di una transazione c per accedere al un generico dataitem nell'ordine k;
- C^{c} carico medio per ogni classe transazionale;

Ognuna di queste formule verrà discussa successivamente nel seguente paragrafo.

Response time di una transazione

Considerando la presenza di T classi transazionali avremo T response time differenti ed un response time medio calcolato come la media pesata sulle frequenze delle singole classi transazionali. In formule sarà:

$$R^{c} = \mathrm{R}_{\mathrm{setup}} + \hat{\mathrm{R}}_{0} + \sum_{k=1}^{M_c} \left(\hat{\mathrm{R}} + \mathrm{R}^{0}_{\mathrm{k}} \right) + \mathrm{R}_{\mathrm{commit}} \qquad (20)$$

e per quanto riguarda il response time medio:

$$R = \frac{\sum_{c=0}^{T} \lambda_c * R^{c}}{\sum_{c=0}^{T} \lambda_c} \qquad (21)$$

Utilizzazione dei set di dataitems

Per quanto riguarda l'utilizzazione dei set di dataitems, la formula (3) scritta nel modello base dovrà essere estesa al caso di più classi transazionali quindi considerando non più soltanto Th_{i} ma $\mathrm{Th}_{\mathrm{i}}^{c}$ ovvero l'utilizzazione del set di dataitems i da parte della classe transazionale c. In formule avremo:

$$\mathrm{Th}_{\mathrm{i}}^{c} = \sum_{k=0}^{M_c} \left[\mathrm{P}_{i,k}^{c} * \left(\sum_{\mathrm{j}=k}^{M_c} \hat{\mathrm{R}} + \sum_{\mathrm{j}=k+1}^{M_c} \tilde{R}_{\mathrm{j}} \right) \right] + \mathrm{R}_{\mathrm{commit}} * \sum_{k=0}^{M_c} \mathrm{P}_{i,k}^{c} \qquad (22)$$

Probabilità di wait

La probabilità di wait rimane legata, come nel modello base, al set di dataitem ed indipendente dalla classe transazione. Per questo, ricordando che come nel modello base la probabilità di wait P_{wi} era l'utilizzazione pesata per la frequenza di accesso, adesso dovremo pesare l'utilizzazione di ogni classe transazionale con la frequenza specifica della classe. In formule avremo:

$$P_{Wi} = \sum_{c=0}^{T} \lambda_c * \mathrm{Th}^{c}_{\mathrm{i}} \qquad (23)$$

Probabilità di contention

Per quanto le probabilità di contention sui vari set di dataitems, queste verranno estese aggiungendo una dipendenza dalla classe transazionale che si vuole considerare. Nello specifico avremo:

- $\hat{P}^{c}_{CONTi,k}$ definita come la probabilità di entrare in conflitto con una transazione di classe c nello stato k cappello, assumendo di voler prendere un lock sul set i;
- $\tilde{P}^{c}_{CONTi,k}$ definita come la probabilità di entrare in conflitto con una transazione di classe c nello stato k tilde, assumendo di voler prendere un lock sul set i;
- $P^{c}_{CONTi,Commit}$ definita come la probabilità di entrare in conflitto con una transazione di classe c nello stato di commit, assumendo di voler prendere un lock sul set i;

L'espressione generale in formule verrà spezzata in quanto non è possibile esprimerla interamente per motivi di spazio. L'espressione del denominatore che rimane comune nelle tre espressioni di P_{cont} è la seguente:

$$Den = \sum_{c=0}^{T} \left\{ \lambda^{c} * \left\{ \begin{array}{l} \sum_{k=0}^{M_c - 1} \left[\sum_{l=0}^{K} P^{c}{}_{i,l} * \hat{R} \right] + \\ \sum_{k=0}^{M_c - 1} \left[\sum_{l=0}^{K} P^{c}{}_{i,l} * \tilde{\tilde{R}}^{c}{}_{l} \right] + \\ \left[\sum_{k=0}^{M_c} P^{c}{}_{i,l} * R_{commit} \right] \end{array} \right\} \right\} \tag{24}$$

Successivamente le tre probabilità saranno:

$$\hat{P}^{c}{}_{CONTi,k} = \frac{\lambda^{c} * \sum_{l=0}^{K} P^{c}{}_{i,l} * \hat{R}}{Den} \tag{25}$$

$$\tilde{\tilde{P}}^{c}{}_{CONTi,k} = \frac{\lambda^{c} * \sum_{l=0}^{K} P^{c}{}_{i,l} * \tilde{\tilde{R}}^{c}{}_{k}}{Den} \tag{26}$$

$$P_{CONTi,Commit} = \frac{\lambda^{c} * \sum_{l=0}^{K} P^{c}{}_{i,l} * R_{commit}}{Den} \tag{27}$$

Tempi di attesa su i set di dataitems

Anche nel calcolo dei tempi di attesa sui set di dataitems, espressione rimane indipendente, come per P_{wi}, dalla classe transazionale. Per questo avremo:

$$\tilde{\tilde{R}}_{Wi} = \sum_{c=0}^{T} \left[\begin{array}{l} \sum_{k=0}^{M_c} \hat{P}^c{}_{CONTi,k} * \left(\frac{\hat{R}}{2} + \sum_{l=k+1}^{M_c} \hat{R} + \sum_{l=k+1}^{M_c} \tilde{\tilde{R}}^c{}_l + R_{commit} \right) + \\ \sum_{k=1}^{M_c} \tilde{\tilde{P}}^c{}_{CONTi,k} * \left(\frac{\tilde{\tilde{R}}_k}{2} + \sum_{l=k+1}^{M_c} \hat{R} + \sum_{l=k+1}^{M_c} \tilde{\tilde{R}}^c{}_l + R_{commit} \right) + \\ P^c{}_{CONTi,Commit} * \frac{\hat{R}}{2} \end{array} \right] \quad (28)$$

Tempi di attesa sugli stati tilde

Il calcolo di $\tilde{\tilde{R}}^c{}_k$, come visto anche nel modello base, è dato dal prodotto di tre fattori che sono $\tilde{\tilde{R}}_{Wi}$, P_{Wi} e $P^c{}_{i,l}$ di cui rispettivamente i primi due non dipendono dalla classe transazionale considerata ma dipendono dal set di dataitems mentre il terzo dipende sia dal set che dalla classe. Questo vuol dire che, data la dipendenza che dovrà necessariamente avere $\tilde{\tilde{R}}^c{}_k$ con lo stato della transazione, si dovrà aggregare rispetto ai set di dataitems, rimanendo la dipendenza dalla classe transazionale. Quanto detto in formule si esprime così:

$$\tilde{\tilde{R}}^c{}_k = \sum_{i=1}^{I} P^c{}_{i,k} * \tilde{\tilde{R}}_{Wi} * P_{Wi} \quad (29)$$

Modello delle risorse hardware

Il modello delle risorse hardware viene leggermente modificato con l'introduzione delle classi transazionali ma non nell'architettura di riferimento bensì nel calcolo del carico sulla CPU. La modifica introdurrà una dipendenza sul carico verso la classe transazionale, che poi andrà a scomparire nel calcolo dell'utilizzazione. In formule avremo:

- Carico della CPU:

$$C^c = I_{Setup} + \sum_{k=0}^{M_c} \left[\left(P^R * I_{READ} \right) + \left(P^W * I_{WRITE} \right) \right] + I_{Commit} \qquad (30)$$

- Utilizzazione CPU:

$$\rho = \frac{\sum_{c=0}^{T} \lambda_c \cdot C^c}{k \cdot MIPS} \qquad (31)$$

Capitolo 4
Validazione del modello analitico

Il procedimento di validazione di un modello analitico consente in primo luogo di verificarne la correttezza rispetto alle ipotesi adottate. La correttezza può essere verificata valutandone l'accuratezza in termini di errori di approssimazione. Da questo punto di vista se gli errori introdotti dal modello rientrano negli intervalli di tolleranza desiderati allora il modello può considerarsi corretto. La validazione consente anche di valutarne la robustezza, ossia di identificare gli intervalli di valori di input entro i quali il modello mantiene i requisiti suddetti. Al fine di valutare gli errori introdotti da un modello analitico sono necessari termini rispetto ai quali confrontare i risultati ottenuti dalla risoluzione numerica. Data la complessità di un sistema transazionale i risultati ottenuti attraverso metodi di misurazione diretta sono inevitabilmente influenzati da fattori relativi all'architettura, all'implementazione e quindi alla qualità dei componenti del sistema stesso. Data, inoltre, la stretta interconnessione che spesso vi è tra i diversi componenti di tali sistemi, non sarebbe oggettivamente realizzabile una valutazione di alcuni componenti isolandone il funzionamento rispetto ad altri. Per tali motivi la validazione dei modelli analitici relativi ai controlli di concorrenza viene effettuata solitamente attraverso modelli simulativi.
In questo lavoro si è proceduto attraverso il suddetto approccio.

4.1 Modello simulativo e descrizione del simulatore

Il modello simulativo adottato per la validazione del modello analitico è illustrato in figura Figura 4-1. In Figura 4-2 è illustrata l'architettura del simulatore utilizzato. Esso è stato costruito basandosi su un DBMS standard. L'architettura consente di intervenire agevolmente nella modifica di alcune caratteristiche. In particolare il modulo relativo al controllo di concorrenza è stato progettato per una facile sostituzione del protocollo da utilizzare. La simulazione è ad eventi discreti [25], I moduli che implementano gli oggetti di simulazione comunicano tra essi attraverso scambi di eventi. Ciò ha consentito di preservare le caratteristiche di indipendenza tra i diversi moduli.

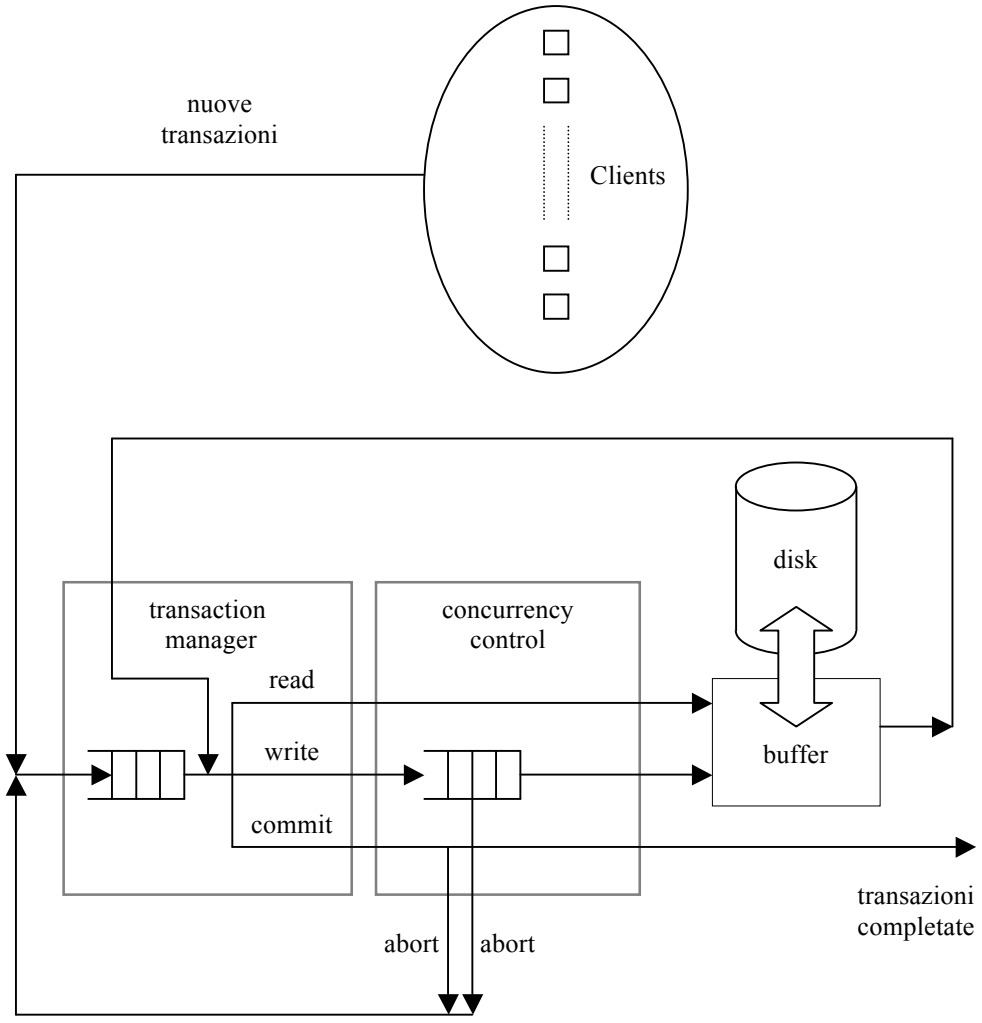


Figura 4-1. Modello simulativo

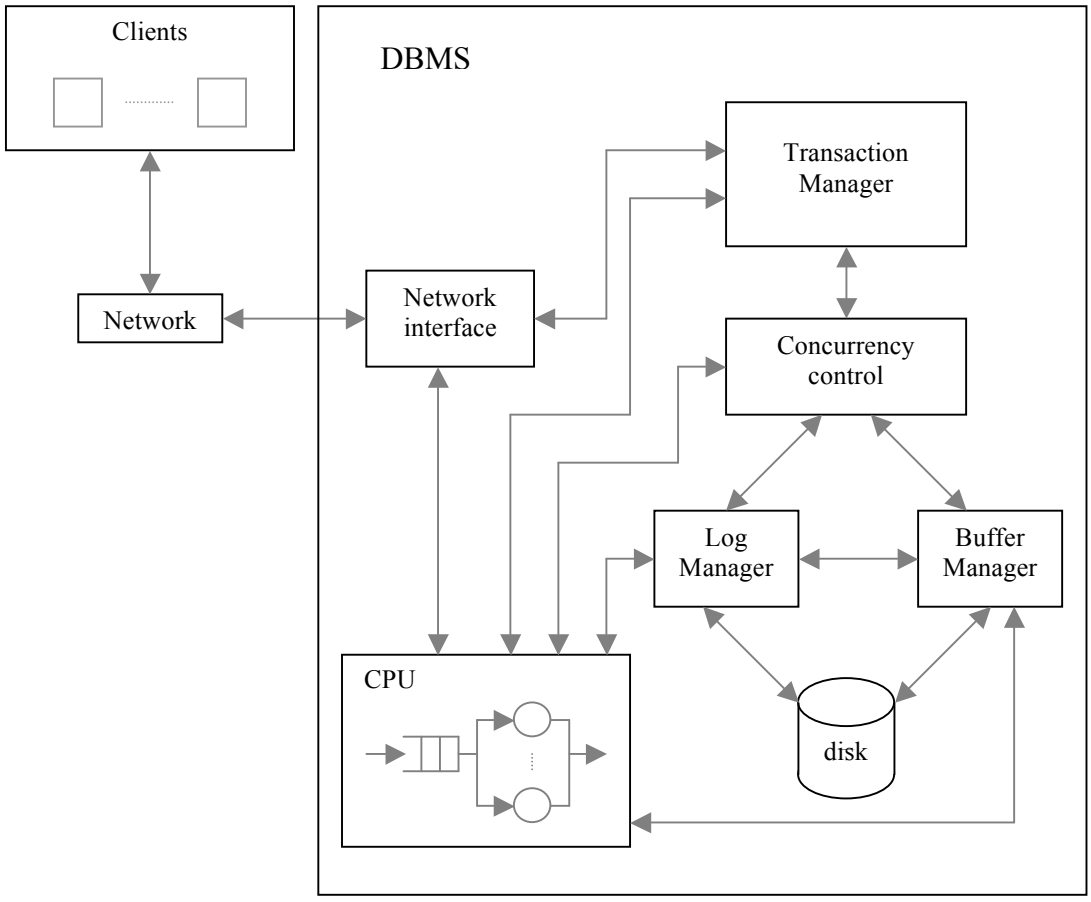


Figura 4-2. Architettura del simulatore

Il workload può essere definito attraverso il Transaction Manager, che dispone di funzioni di distribuzione per la generazione delle transazioni configurabili attraverso alcuni parametri. Tra questi parametri vi sono, ad esempio, la lunghezza delle transazioni in termini di operazione effettuate, la distribuzione tra letture e scritture e la distribuzione degli accessi sui dataitems. L'esecuzione delle transazioni è 'triggerata' dal modulo Client,. Nel caso di sistema chiuso tale modulo simula la presenza di un numero stabilito di clients che generano le richieste secondo una distribuzione del relativo think time. Nel caso di sistema aperto il modulo Client genera le richieste secondo una distribuzione dei tempi di interarrivo. L'esecuzione dei reruns delle transazioni è generata secondo una distribuzione di tempi di backoff. Attraverso altri parametri si possono configurare alcune caratteristiche del sistema,

come la dimensione della base di dati, la dimensione del buffer, il numero di CPU e la relativa potenza di elaborazione. La politica di gestione dei dati sul buffer è di tipo *least recently used*. Infine ogni tipo di richiesta relativa alla CPU o al disco è configurabile rispettivamente in termini di numero di istruzioni macchina e di tempi di accesso in lettura o scrittura.

4.2 Validazione del modello analitico

In questo paragrafo si presentano i risultati ottenuti dal modello analitico confrontandoli con quelli ottenuti dai test simulativi. Per ogni test viene fornita la configurazione adottata ed i risultati ottenuti vengono illustrati graficamente.

4.2.1 Pattern di accesso ai dati uniforme

Di seguito verranno illustrati gli andamenti dei parametri critici per lo studio delle performance dei sistemi transazionali confrontando i risultati ottenuti dal calcolo numerico del modello analitico illustrato con quelli del simulatore utilizzato.
In prima istanza verrà presentato un workload costituito da un pattern di accesso ai dati uniforme, ovvero una situazione in cui le classi transazionali hanno la stessa probabilità di accedere al set di dataitems i nell'ordine k, per ogni i e k. In termini pratici la matrice di probabilità $P^{c}_{i,k} = 1/I \quad \forall i \in I, \forall k \in M$. Tutti i parametri saranno rappresentati rispetto alla frequenza di arrivo delle transazioni con la quale è stata fatta la simulazione. I parametri che saranno rappresentati sono:

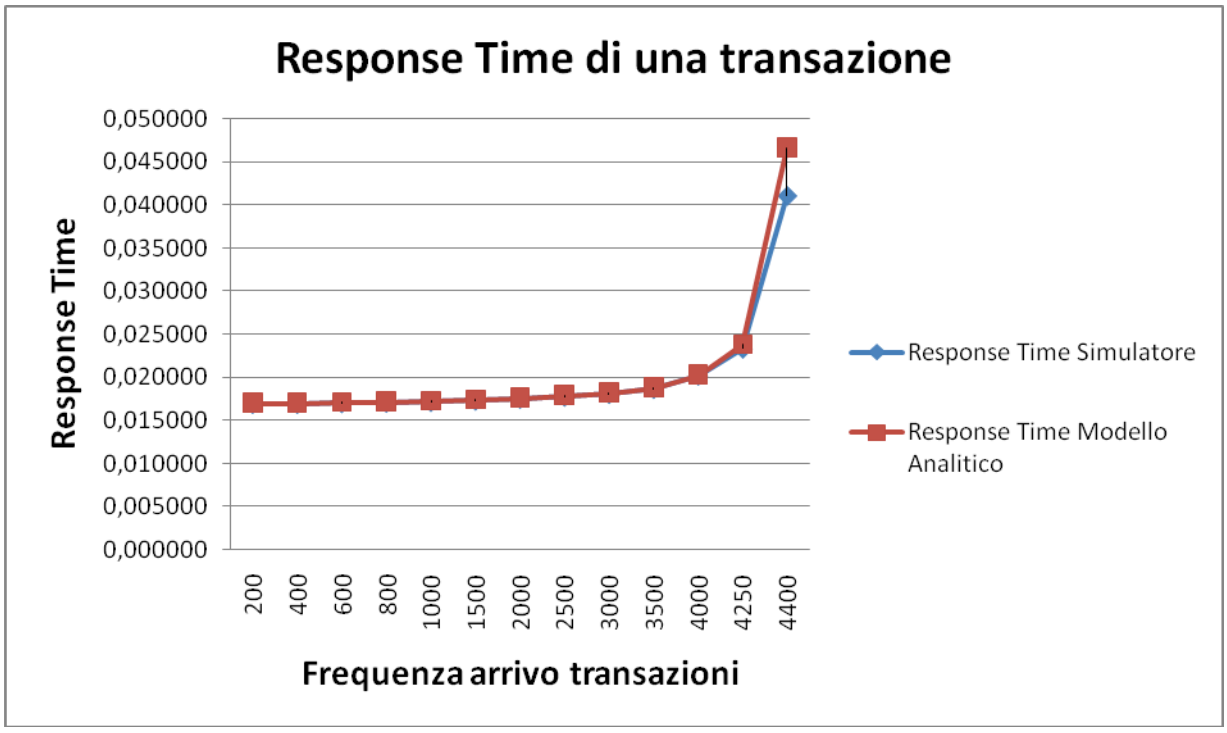
- Response time medio di una transazione;
- Utilizzazione dei singoli set di dataitems;
- Durata dei locks sui set di dataitems
- Tempo di attesa negli stati $\tilde{\tilde{R}}$.

Tutti in funzione della frequenza λ di arrivo delle transazioni.

Response Time di una transazione

Nel Grafico 1 viene mostrato l'andamento del response time delle transazioni.

Grafico 1



La configurazione del carico con la quale è stato condotto il test è la seguente:

T	1
I	5.000
M	5
I_{Setup}	300.000
I_{Write}	50.000
I_{Read}	50.000
I_{Commit}	350.000
N_{CPU}	4
MIPS	1.000.000.000
T_{IO}	0.004

Si può osservare come il comportamento del modello è praticamente identico in condizioni di carico accettabile rispetto al simulatore. Questo risultato ci permettere di apprezzare la validità del modello analitico.

Le due curve cominciano ad assumere valori leggermente discostanti soltanto nel momento immediatamente precedente al trashing del simulatore. Questa è una situazione tipica, infatti i risultati ottenuti in prossimità del punto di saturazione non sono da considerarsi attendibili per eventuali confronti per studi comparativi. In questo caso, con la configurazione del carico riportata in tabella, il punto di saturazione si trova tra la frequenza di 4400 e 4550, quindi l'ultimo valore utile del nostro esperimento si avvicina

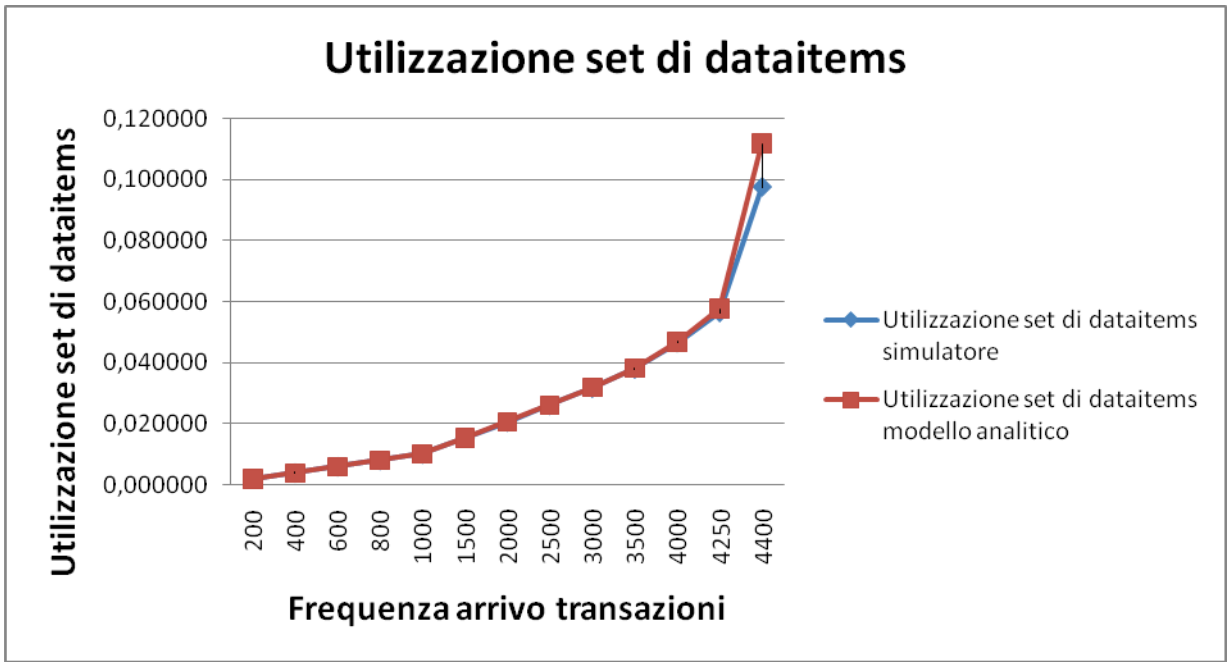
molto e come è possibile vedere dal grafico è il valore che si discosta maggiormente dal simulatore.

L'andamento del grafico è tipico dei tempi di risposta nei sistemi transazionali, in cui il valore cresce uniformemente con un aumento moderato del carico fino ad avvicinarsi al punto di saturazione in cui il valore tende ad esplodere seguendo un andamento esponenziale.

Utilizzazione dei dataitems o Probabilità di Wait

Nel grafico 2 viene riportata la probabilità di entrare in uno stato di wait a causa di una richiesta di lock, anche definita come utilizzazione dei set di dataitems. Dal modello si ricava la suddetta probabilità in funzione del set di dataitems toccato ma, considerando l'uniformità di accesso ai dati scelta per questa prima parte di test, il valore della probabilità sarà identico per tutti i set di dataitems.

Grafico 2



La configurazione del carico con la quale è stato condotto il test è la seguente:

T	1
I	5.000
M	5
I_{Setup}	300.000
I_{Write}	50.000
I_{Read}	50.000
I_{Commit}	350.000
N_{CPU}	4
MIPS	1.000.000.000
T_{IO}	0.004

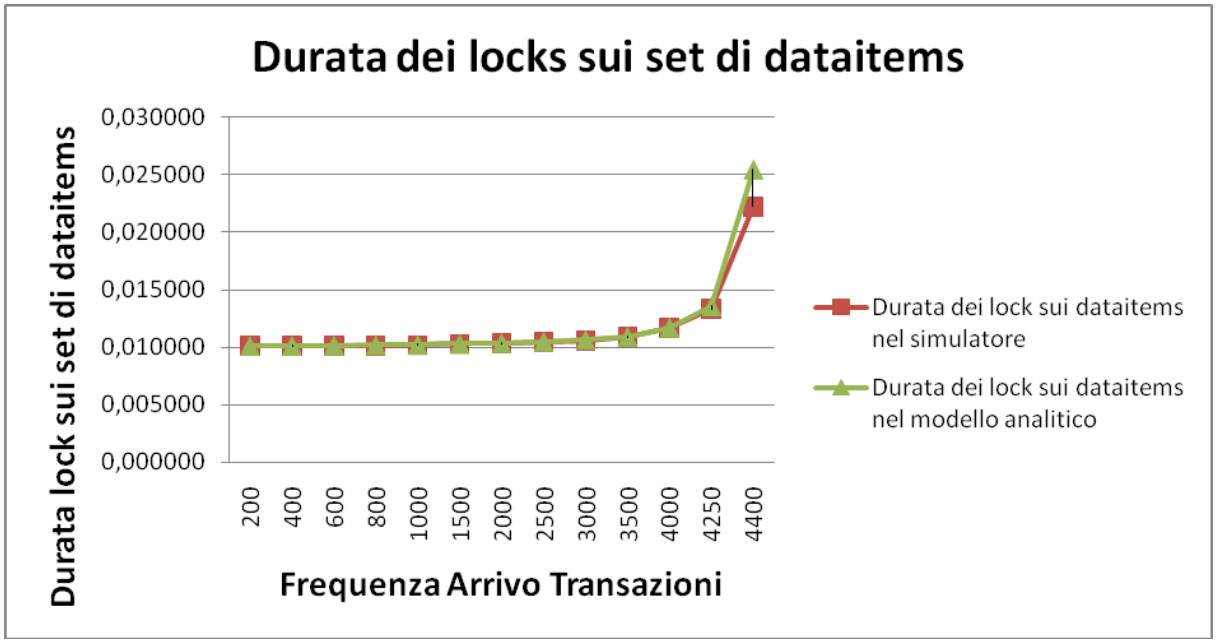
Anche in questo caso il modello risulta aderente al simulatore che ne conferma quindi la validità.

Per gli stessi motivi illustrati precedentemente, anche in questo caso i valori intorno al punto di saturazione risultano più distanti delle media delle distanze ottenute in condizioni di carico normale. La curva è paragonabile all'andamento riscontrato nel Grafico 1 in quanto più il carico aumenta e maggiore è la possibilità che una transazione riacceda agli stessi set di dataitems già acceduti in precedenza.

Durata dei lock sui set di dataitems

Nel grafico 3 viene illustrato la durata media di un lock su un generico set di dataitems. Come per l'utilizzazione anche la durata dipende dal set di dataitems considerato ma utilizzando le stesse premesse fatte nel paragrafo precedente, ci si accorge che questo valore è identico per tutti i set(caso di pattern di accesso uniformi).

Grafico 2



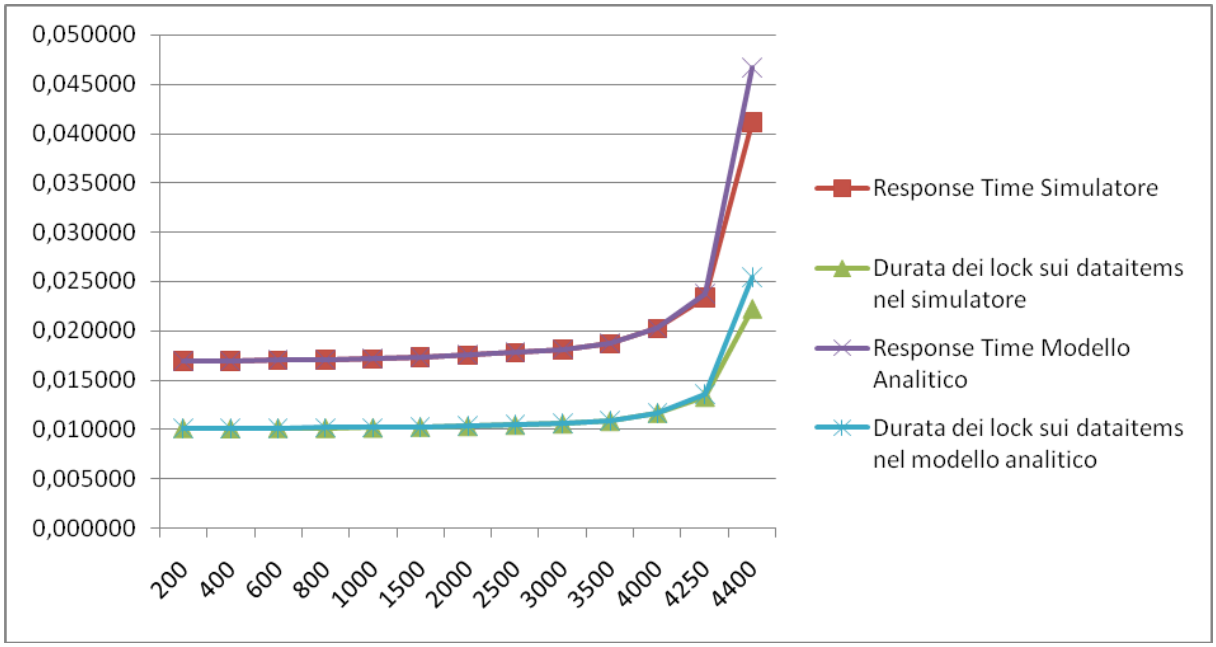
La configurazione del carico con la quale è stato condotto il test è la seguente:

T	1
I	5.000
M	5
I_{Setup}	300.000
I_{Write}	50.000
I_{Read}	50.000
I_{Commit}	350.000
N_{CPU}	4
MIPS	1.000.000.000
T_{IO}	0.004

Oltre a ribadire l'aderenza del modello analitico rispetto al simulatore, una cosa interessante è rappresentata dal confronto di questo andamento con quello del response time riportato nel grafico 1. Questo andamento è riportato nel Grafico 4.

Dal confronto delle curve si può apprezzare come la curva della durata media dei locks segua quella del response time delle transazioni pur rimanendo costantemente sotto. Sarebbe stato chiaramente un errore osservare un tempo di durata dei lock maggiore della durata della transazione.

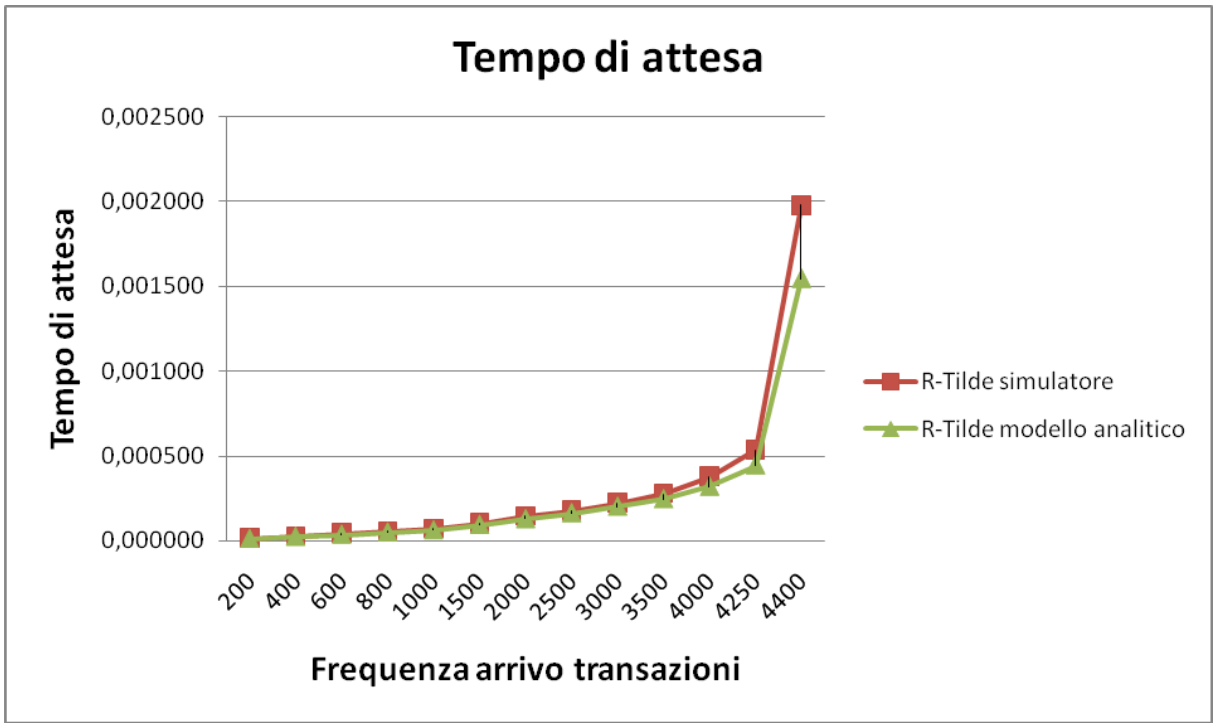
Grafico 4



Tempi di attesa sugli stati tilde

Nel grafico 5 verrà riportato l'andamento del valore di $\tilde{\tilde{R}}$. Questo valore dipendeva dallo stato di esecuzione della transazione, ma considerando il pattern di accesso uniforme anche questo valore rimane costante per tutti gli stati. Comprendere questa uniformità è abbastanza semplice data la formula del calcolo $\tilde{\tilde{R}}_k$ e considerando le deduzioni fatte nei paragrafi precedenti per utilizzazione e la durata dei lock.

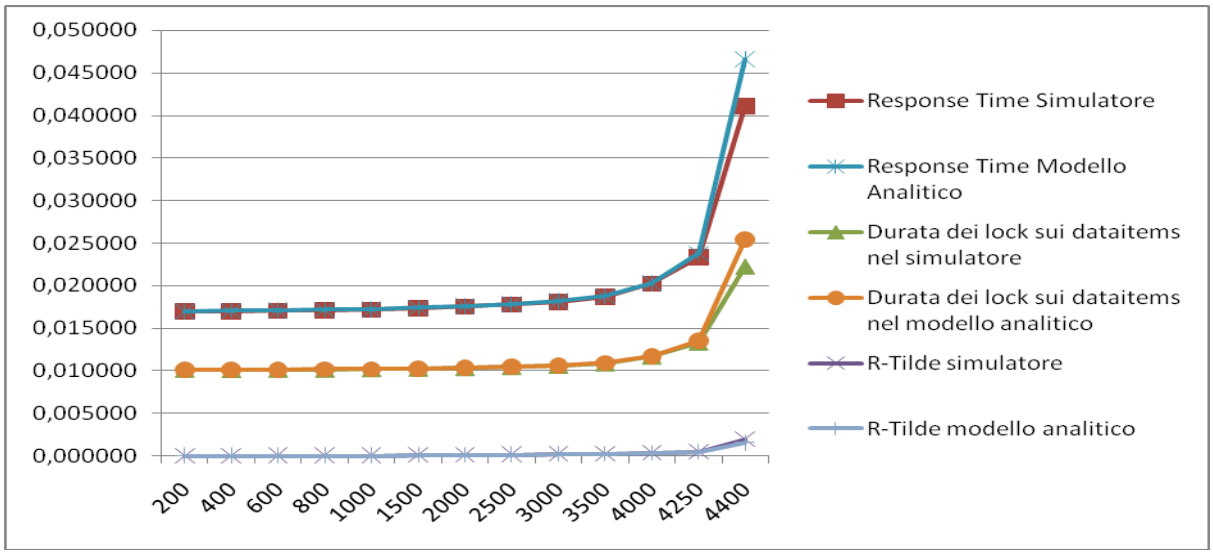
Grafico 5



In questo grafico si nota come il modello tende ad allontanarsi avvicinandosi al punto di saturazione con un errore relativo maggiore rispetto a quello osservato negli altri grafici. I valori riportati nel grafico sono molto più bassi di quelli riportati nei grafici precedenti ma c'è da considerare il fatto che $\tilde{\tilde{R}}$ è il tempo di attesa speso negli stati tilde, quindi deve essere necessariamente molto più basso dei precedenti.

Nel Grafico 6 mostriamo un riepilogo dei tre parametri per cui ha senso un confronto sulla base delle stesse scale di misura.

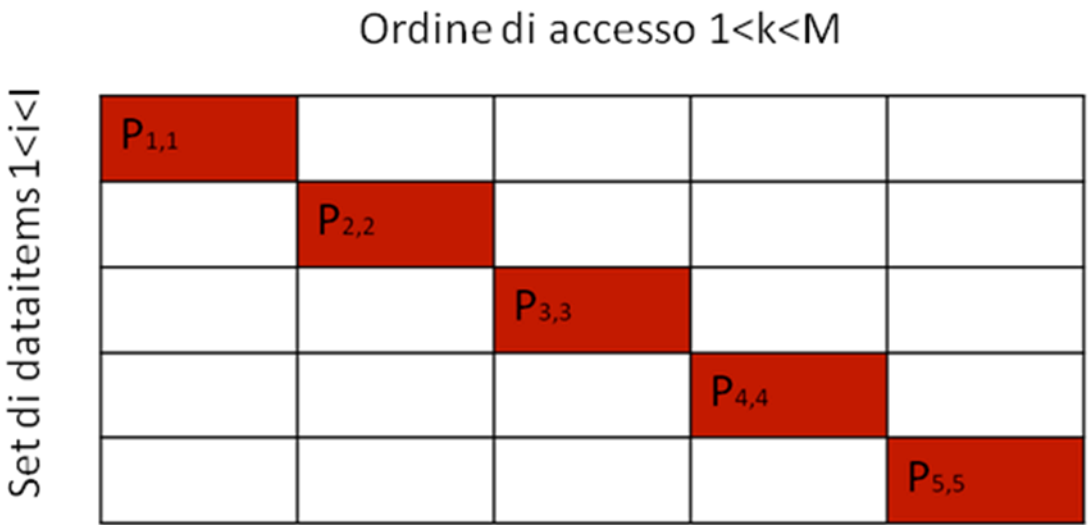
Grafico 6



4.2.2 Pattern di accesso ai dati non uniforme

Dopo aver fatto un'analisi considerando un pattern di accesso uniforme ai set da parte delle transazioni, verrà modificato proprio quest'ultimo utilizzando un patter di accesso a blocchi. Per patter di accesso a blocchi si intende una maniera per veicolare l'accesso a determinati set di dataitems sulla base dello stato di esecuzione in cui si trova la transazione. Il carico è stato modificato in quanto il calcolo dei valori, che nel caso di pattern uniformi risultavano uguali per ogni set, adesso devono essere calcolati utilizzando il valore medio e per rendere più attendibile il risultato il carico è stato aumentato. Come si potrà notare dall'analisi dei grafici riportati il punto di saturazione è cambiato rispetto ai test fatti nel paragrafo precedente. Questa variazione è dovuta al fatto che le transazioni, con questo nuovo patter di accesso, accedono ad un ridotto set di dataitems nello stesso ordine quindi la probabilità di contention è sicuramente maggiore e l'aumento del rate di arrivo delle transazioni comporta un notevole incremento del carico fino ad arrivare alla salutazione.

L'implementazione di questo pattern di accesso ricade nella personalizzazione della matrice di probabilità $P^{c}_{i,k}$

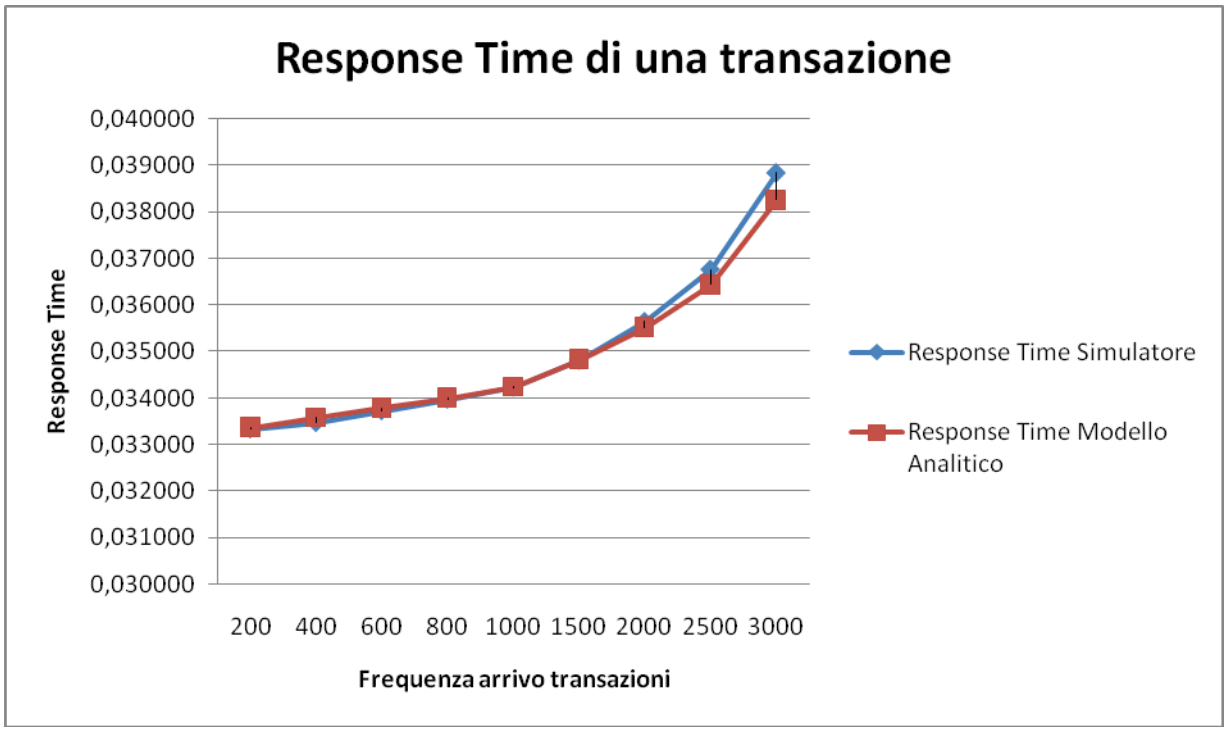


Da questa matrice si legge chiaramente come, per esempio, le transazioni in escuzione nello stato 3 cappello potranno richiedere soltanto dataitems all'interno del set 3.

Response Time di una transazione

Nel Grafico 7 viene mostrato l'andamento del response time delle transazioni.

Grafico 7



La configurazione del carico con la quale è stato condotto il test è la seguente:

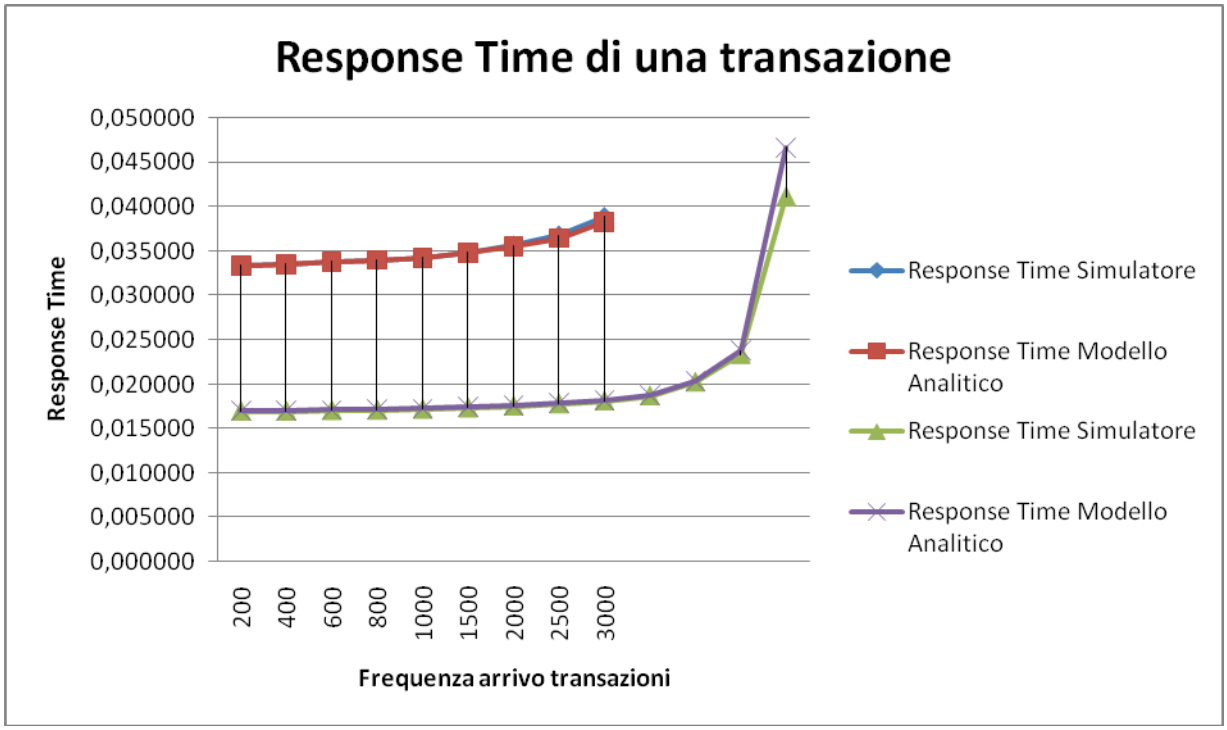
T	1
I	20.000
M	10
I_{Setup}	300.000
I_{Write}	50.000
I_{Read}	50.000
I_{Commit}	350.000
N_{CPU}	4
MIPS	1.000.000.000
T_{IO}	0.004

Si può osservare come il comportamento del modello segue abbastanza il simulatore pur rimanendo costantemente sotto la curva. Avvicinandosi poi verso la saturazione, ovvero intorno alla frequenza di 3500 TPS, le due curve cominciano a distanziarsi ma il gap è decisamente accettabile e comprensibile considerando la casualità introdotta dal simulatore e la vicinanza alla saturazione. Un indice per valutare la bontà del modello è sicuramente l'errore relativo commesso nel punto precedente alla saturazione a 3000TPS. Questo rimane inferiore al 1,5% quindi considerato accettabile nei limiti di validità del modello.

Possiamo osservare come in condizioni di carico non uniforme la tendenza della curva ad avere un comportamento esponenziale con l'aumento della frequenza delle transazioni è più accentuata.

Nel Grafico 8 vengono paragonati i risultati ottenuti rispetto al pattern di accesso uniforme.

Grafico 8

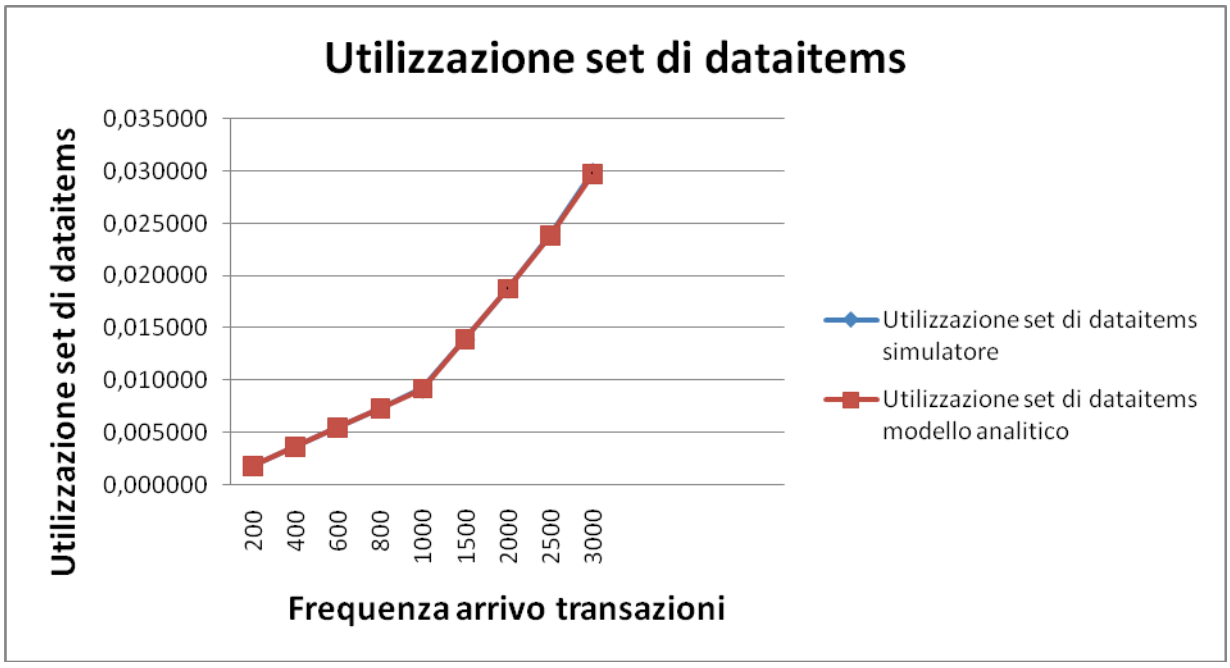


Questo grafico è la dimostrazione dell'effetto della caratterizzazione dei pattern di accesso ai dati. Chiaramente con pattern uniforme il response time rimane molto sotto quello raggiunto con pattern a blocchi visto l'aumento considerevole della probabilità di contention sui set di dataitems. Si nota anche lo spostamento del punto di saturazione nelle due situazioni.

Utilizzazione dei dataitems o Probabilità di Wait

Nel grafico 9 viene riportata l'utilizzazione dei set di dataitems. Il valore graficato è una media su tutti i set di dataitems della base di dati.

Grafico 9



La configurazione del carico con la quale è stato condotto il test è la seguente:

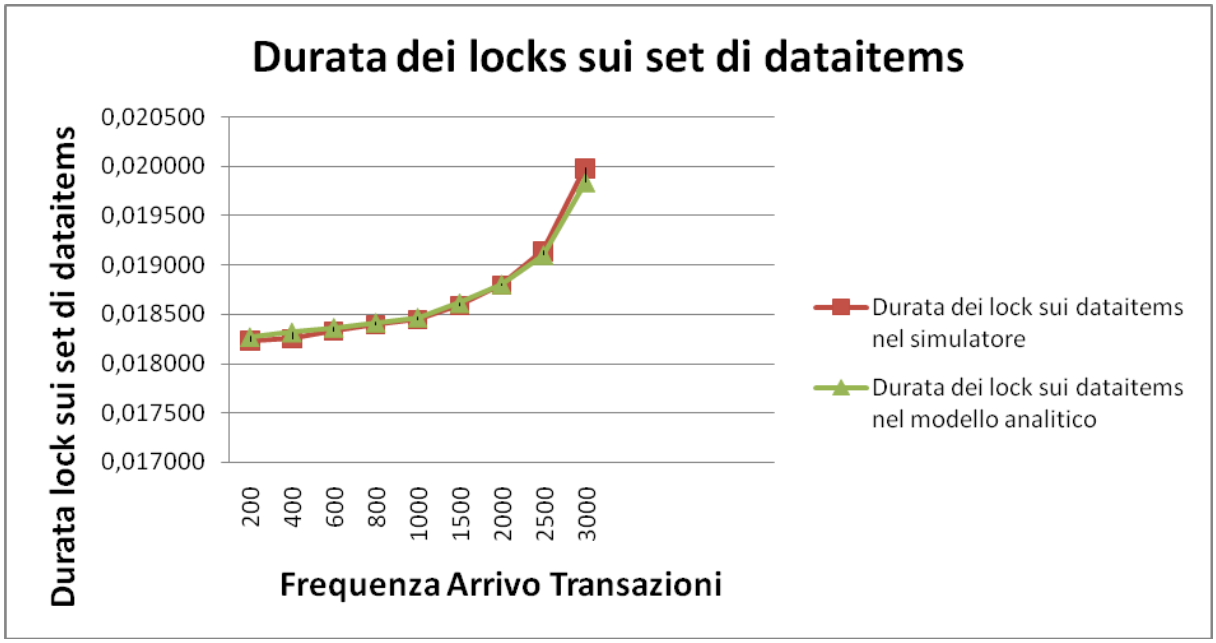
T	1
I	20.000
M	10
I_{Setup}	300.000
I_{Write}	50.000
I_{Read}	50.000
I_{Commit}	350.000
N_{CPU}	4
MIPS	1.000.000.000
T_{IO}	0.004

Il modello è estremamente vicino al simulatore e rimane caratteristica la forma della curva dovuta al fatto che la probabilità di wait è direttamente proporzionale sia alla frequenza lambda che al tempo di attesa dei locks. Con questo pattern di accesso la probabilità di contention aumenta drasticamente con l'aumentare della frequenza di arrivo delle transazioni e questo giustifica la forte impennata dell'utilizzazione ad alti rate di arrivo.

Durata dei lock sui set di dataitems

Nel grafico 10 viene illustrato la durata media di un lock su un generico set di dataitems. Come per l'utilizzazione anche questo rappresenta un valore medio rispetto al totale dei set di dataitems.

Grafico 10

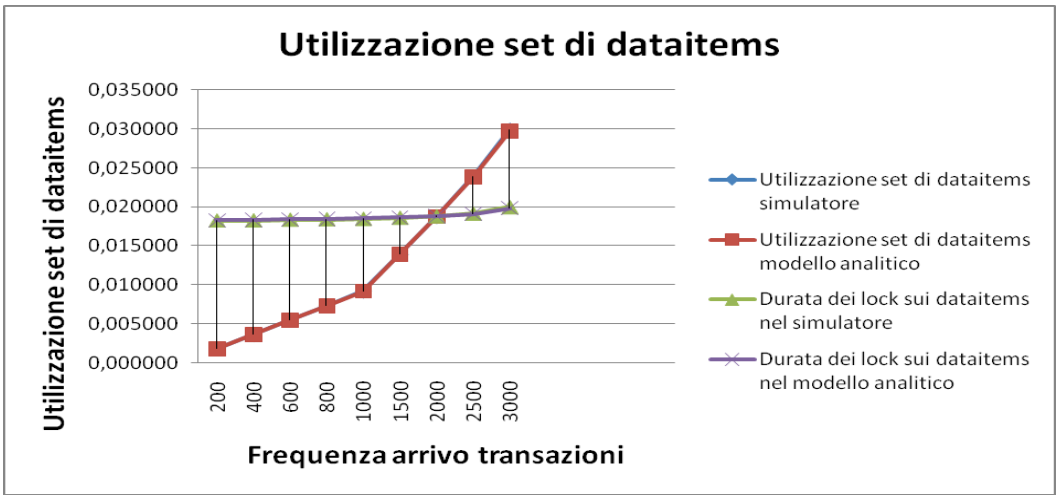


La configurazione del carico con la quale è stato condotto il test è la seguente:

T	1
I	20.000
M	10
I_{Setup}	300.000
I_{Write}	50.000
I_{Read}	50.000
I_{Commit}	350.000
N_{CPU}	4
MIPS	1.000.000.000
T_{IO}	0.004

Il modello analitico è molto vicino al simulatore anche intorno al punto di saturazione. Nel grafico 11 viene mostrato in dettaglio il motivo della forma del grafico 9. Infatti se consideriamo la proporzionalità diretta che c'è tra l'utilizzazione, la durata dei lock e la frequenza, ogni punto del grafico 11 è da considerarsi come il prodotto tra la frequenza delle transazioni, riportata nelle ascisse, e il valore del tempo di lock.

Grafico 11

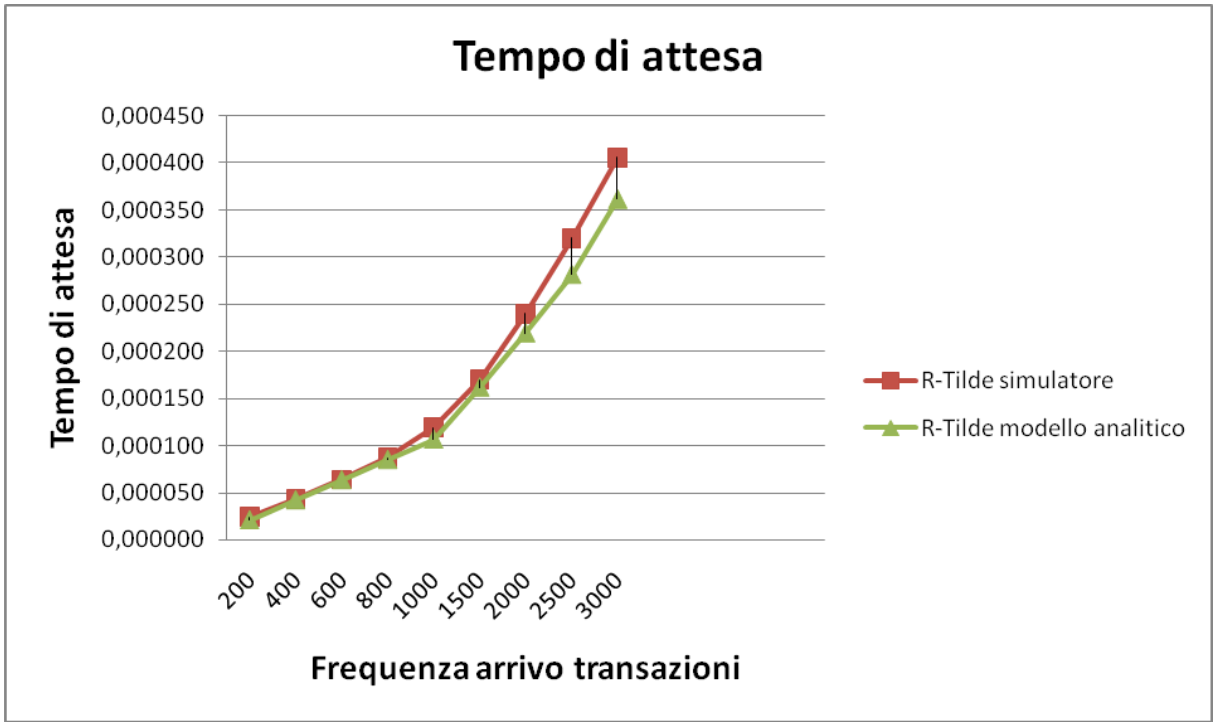


Tempi di attesa sugli stati tilde

Nel grafico 12 verrà riportato l'andamento del valore di $\tilde{\tilde{R}}$. Questo valore dipende, come mostrato nel modello, dallo stato di esecuzione della transazione quindi è stato considerato il valore medio rispetto alla lunghezza della transazione. I valori assunti dal simulatore e dal modello sono simili nella situazione di sistema a carico standard e si discostano leggermente quando il carico comincia a saturare il sistema fino ad arrivare nel punto di saturazione ad un errore relativo del 10%.

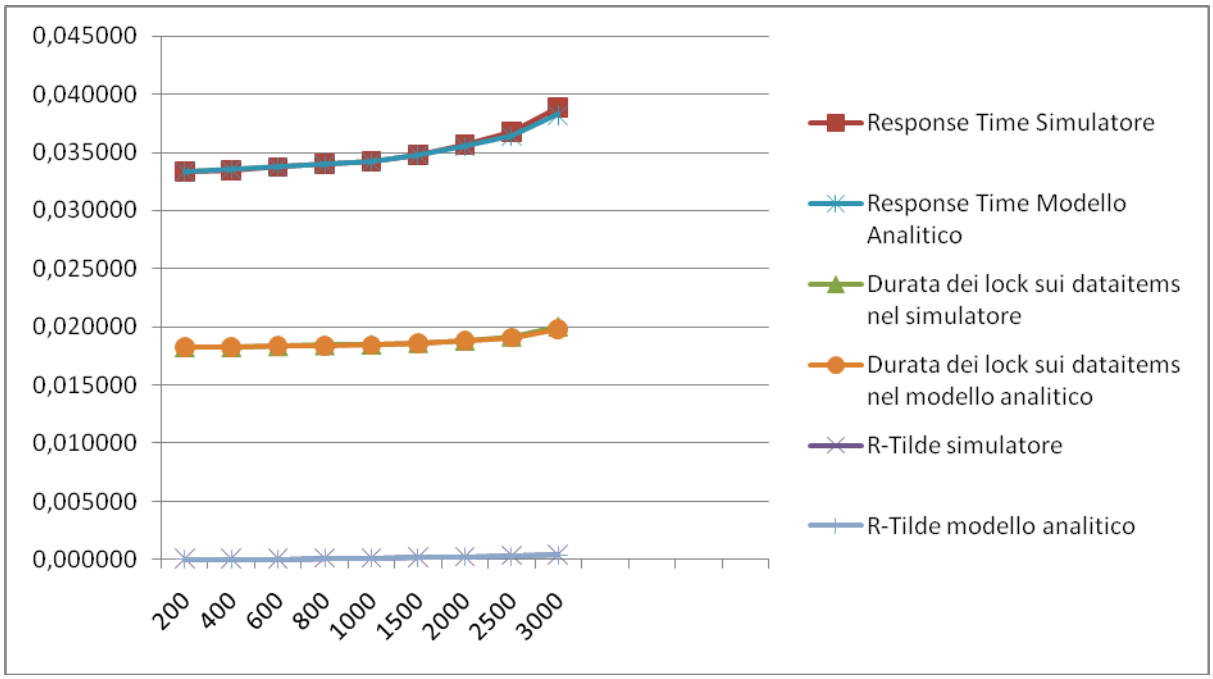
Una causa di questa leggera divergenza potrebbe essere la non validità dell'ipotesi fatta durante l'evoluzione del modello con la quale si restringeva il numero di transazioni in attesa per un lock ad una soltanto. In situazioni di alta contention (tipiche della saturazione) non ci sono più le condizioni per rendere valida questa ipotesi ed è per questo che il valore del tempo di attesa si discosta da quello del simulatore.

Grafico 12



Nel grafico 13 vengono riportati i risultati aggregati per la trattazione dei patter di accesso non uniformi.

Grafico 13



4.2.3 Pattern di accesso ai dati non uniforme con presenza di più classi transazionali

Si presenta ora il risultato di un test effettuato per validare il modello analitico esteso con la presenza di 2 classi transazionali. Per rendere interessante il risultato è stato caratterizzato in una maniera particolare sia il carico di lavoro che il patter di accesso. In particolare per effettuare questi test è stato utilizzato un carico simile a quello utilizzato per la prova con i pattern di accesso non uniformi del paragrafo precedente ma con la differenza della grandezza del buffer che fa da cache per l'accesso al disco. Infatti impostando un buffer simile a quello utilizzato precedentemente si correva il rischio di far rimanere nel buffer i set di dataitems più richiesti rispettando il pattern impostato e falsando la durata dei lock che è preponderante per il calcolo del response time. Quanto detto verrà compreso meglio durante la spiegazione del pattern di accesso utilizzato.

Il patter di accesso utilizzato è ovviamente diverso per ogni classe transazionale individuata. Schematizzando si avrà:

- Classe transazionale 1: Patter di accesso a blocchi utilizzato anche nel paragrafo precedente;

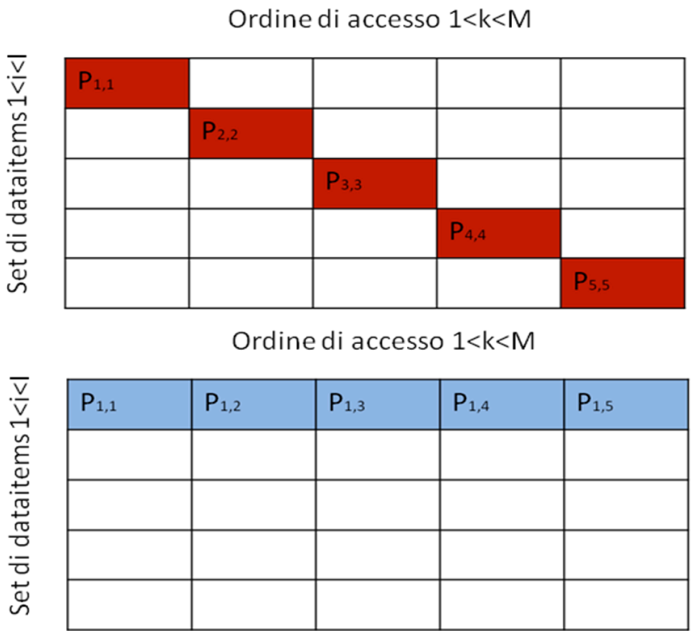
- Classe transazionale 2: Patter di accesso che restringe il numero di set eccedibili dalla transazione nel primo quindi di quelli disponibili. Nel test avremo 5000 set totali quindi la classe 2 potrà accedere soltanto ai primi 1000 set nel database, senza limitazione sull'ordine;

Adesso si può apprezzare il motivo per cui è stata modificata la grandezza del buffer. Nei precedenti test, il buffer era grande il 20% dei set che nel caso in questione sarebbe 1000 entry. Se analizziamo i patter appena elencati, è evidente che i primi 1000 set saranno quelli maggiormente acceduti in quanto la classe transazionale 2 richiede soltanto quelli e la classe transazionale 1 richiede quei stessi set durante il primo stato di esecuzione.

Dai test effettuati con buffer 20% il risultato atteso si è completamente ribaltato in quanto, come vedremo successivamente, il response time ci si attende maggiore è sicuramente quello legato alla classe transazionale 2 in quanto per le transazioni che ne fanno parte, essendo ristretto il numero di set di dataitems a cui possono accedere, hanno un'altissima probabilità di contention e da questo ne segue un response time maggiore. Questo risultato viene falsato dalla presenza di almeno l'80% dei primi 1000 set di dataitems nel buffer e la durata dei locks in caso di dataitems presenti in memoria e non sul disco è minore di alcuni ordini di grandezza. Quindi la probabilità di contention è sicuramente molto più alta nelle transazioni di classe 2 ma la durata dei locks è molto inferiore rispetto ai locks richiesti dalle transazioni di classe 1 e questo tende a livellare i due response time.

Per evitare questo si è deciso di impostare un buffer grande esattamente quanto il numero di set di dataitems nella base di dati e questo elimina il fenomeno appena descritto.

In figura viene mostrata la maniera con cui verranno implementati i pattern di accesso sopra descritti:

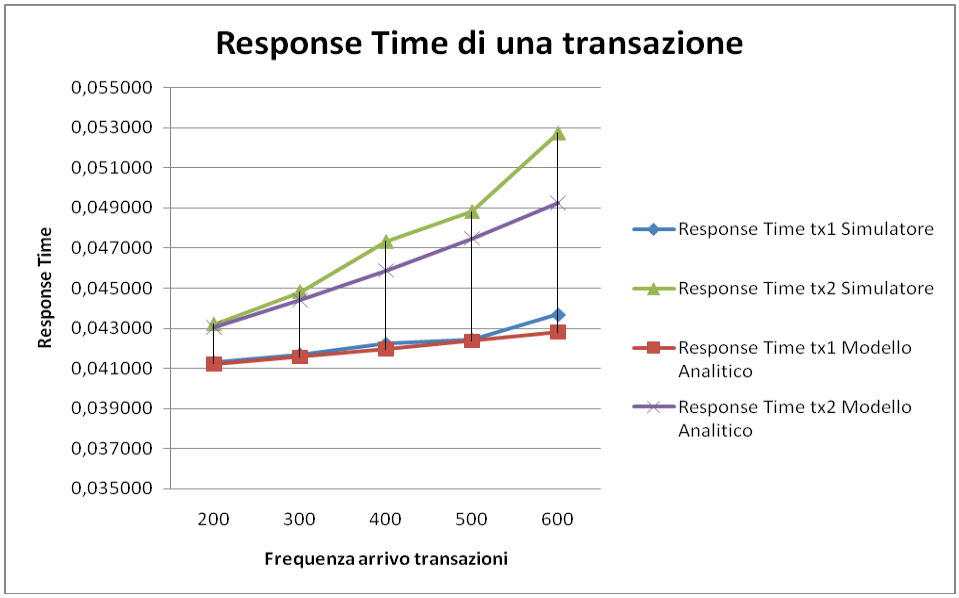


Data la presenza di più classi transazionali, è necessario indicare la frequenza di arrivo e la lunghezza per ogni calsse transazionale. Per semplicità verrà considerata una sola frequenza di arrivo che deve essere interpretata come uguale per tutte le calssi transaionali. Stessa assunzione verrà fatta per la lunghezza delle classi.

Response Time di una transazione

Nel Grafico 14 viene mostrato l'andamento del response time di entrambi le classi transazionali.

Grafico 14



La configurazione del carico con la quale è stato condotto il test è la seguente:

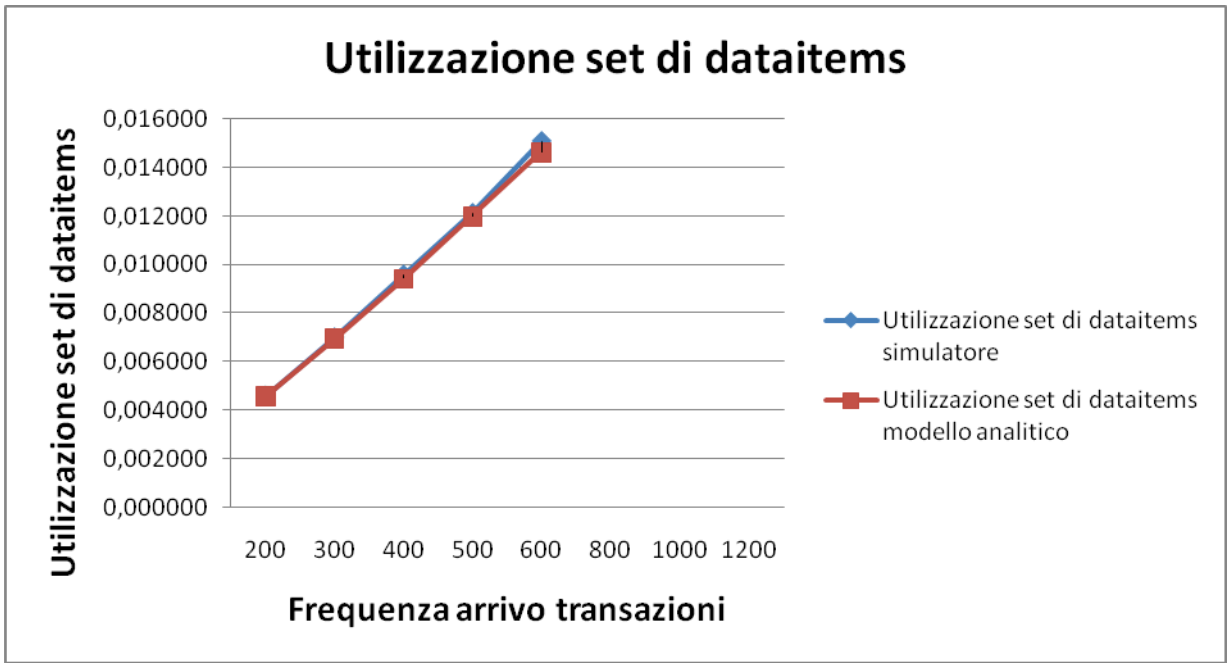
T	2
I	20.000
M	10
I_{Setup}	300.000
I_{Write}	50.000
I_{Read}	50.000
I_{Commit}	350.000
N_{CPU}	4
MIPS	2.000.000.000
T_{IO}	0.004

Si può osservare come il response time della seconda classe transazionale sia decisamente maggiore rispetto a quello della prima classe e questo era ipotizzabile in quanto la classe 2 è veicolata all'utilizzo soltanto dei primi 1000 set di dataitems che sono inoltre acceduti nei primi stati di esecuzione anche dalla prima classe transazionale. Questo comporta un aumento dei tempi di durata dei lock e dell'utilizzazione, riflettendosi alla fine sul response time.

Il modello risulta molto preciso nel caso della prima classe transazionale ma si discosta leggermente dalla curva della seconda classe. L'errore relativo riscontrato in saturazione è del 7% quindi ampiamente accettabile considerando la situazione di alta contention in cui il sistema si trova con questo tipo di pattern di accesso. Altro motivo potrebbe essere rappresentato dalla presenza dell'ipotesi introdotta durante la modellazione che limita ad 1 le transazioni in attesa su un dataitem. Aumentando il carico ed il pattern, nelle zone in cui le due curve si allontanano è sicuramente presente un'alta probabilità di contention e quindi il possibile verificarsi del fenomeno appena descritto.

Utilizzazione dei dataitems o Probabilità di Wait

Grafico 15



Nel grafico 15 viene riportata l'utilizzazione dei set di dataitems. Il valore graficato è una media su tutti i set di dataitems e su tutte le classi transazionali. Come si può osservare il modello è estremamente vicino al simulatore.

4.3 Osservazioni finali

Questo capitolo è stato scritto con lo scopo di validare il modello analitico presentato nel capitolo 3 ed evidenziare contestualmente alcuni comportamenti caratteristici dovuti all'assunzione di patter di accesso ai dati non uniformi.

Lo sviluppo del simulatore per la validazione è avvenuto attraverso un processo iterativo che ha contribuito al miglioramento del modello analitico, raffinandolo in alcuni contesti in cui risultava troppo impreciso. I tre filoni di test:

- Pattern uniforme;
- Pattern non uniforme;
- Pattern non uniforme con multi classe;

sono risultati abbastanza attendibili considerando il valore dell'errore relativo costantemente sotto il 4% nei primi due casi e inferiore al 10% nel terzo caso. Un grafico molto caratteristico è sicuramente il numero 8 in quanto all'interno vengono messi a confronto i response time dei differenti pattern di accesso e si nota come, a parità di carico, la specializzazione del pattern comporta una radicale modifica ai temi di risposta, sia dal punto di vista del valore finale che mediante la modifica di alcuni parametri importanti come per esempio il limite di saturazione.

Sulla base di quanto appena detto consideriamo i risultati ottenuti accettabili dato lo scopo di questa modellazione ovvero far risaltare la differenza in termini di comportamento di un sistema transazionale variando il pattern di accesso si dati scelto.

Conclusioni e sviluppi futuri

Il quadro relativo ai protocolli per il controllo della concorrenza nei sistemi transazionali presentato nel Capitolo 1 illustra le diverse strategie che sono state definite e come vengono adottate dai protocolli. Gli effetti sulla performance dei sistemi transazionali dovuti alla presenza dei controlli di concorrenza sono principalmente riscontrabili attraverso un degrado delle prestazioni a cui sono soggetti. D'altra parte, la garanzia delle cosiddette proprietà *ACID* non può essere a carico degli utenti di un sistema transazionale, ma deve essere parte integrante delle funzionalità che un tale sistema deve fornire. A tal fine la scelta delle diverse strategie di gestione della concorrenza da implementare nel protocollo utilizzato costituiscono un fattore critico sin dalle prime fasi di progettazione di un sistema transazionale. Come mostrato dalla maggior parte dei lavori di modellazione e valutazione che fornisce la letteratura scientifica, l'impatto che ognuna di queste strategie può avere sulla performance del sistema dipende da una molteplicità di fattori. Una tale valutazione dunque non può limitarsi all'analisi dei motivi per i quali una strategia sia responsabile dell'incremento del carico di elaborazione e dei tempi di risposta delle transazioni, così come illustrato nel Capitolo 2, ma è necessario condurre un processo di valutazione approfondito che tenga conto anche delle caratteristiche del sistema e del carico in cui la strategia viene impiegata.

Lo scenario applicativo di un sistema transazionale deve costituire un altro parametro di valutazione. In tal senso nel Capitolo 2 si è parlato della caratterizzazione del workload e dei relativi modelli di riferimento. Tali osservazioni hanno spinto in questo lavoro ad introdurre più classi transazionali, ai fini di una modellazione più accurata del workload, e pattern di

accesso ai dati non uniformi, per catturare l'influenza dell'ordine di accesso sui tempi di risposta.

Come già detto un processo di valutazione non può prescindere da alcuni fattori legati alle caratteristiche del sistema in esame, tra cui la disponibilità di risorse. Un'analisi priva di tali considerazioni porterebbe a conclusioni che, come mostrato da alcuni studi presentati in letteratura, non catturerebbero l'effettivo comportamento del sistema dovuto alle interazioni tra i controlli di concorrenza e le risorse disponibili. Inoltre ciò porterebbe a valutazioni poco realistiche nell'effettuare un confronto tra diverse strategie per il controllo della concorrenza.

Le precedenti osservazioni forniscono un'idea della complessità legata ai processi di valutazione della performance dei sistemi transazionali. I metodi e gli strumenti presentati nel Capitolo 2 costituiscono validi punti di riferimento per tali processi, sia perché spingono verso una formalizzazione dei problemi da trattare, sia perché sono largamente testati ed affidabili. In relazione a questi aspetti però non bisogna perdere di vista le assunzioni che vengono fatte ai fini del relativo utilizzo e le approssimazioni che tali assunzioni introducono nella modellazione dei sistemi reali. Per questi motivi la modellazione e la valutazione dei sistemi devono essere sempre associate a processi di identificazione e quantificazione degli errori che si possono commettere.

Di modelli analitici per l'analisi del protocollo Strict 2PL ed in generale del 2PL in letteratura scientifica ne esistono vari ed ognuno coglie alcune sfaccettature non approfondite da altri ma nessuno ha mai sviluppato il modello considerando i pattern di accesso ai dati. In altre parole la specializzazione innovativa di questo lavoro è la caratterizzazione dell'ordine con il quale le varie classi transazionali accedono ai dataitems della base di dati. Come ipotizzato e poi successivamente consolidato durante la validazione, questo tipo di caratterizzazione del workload sposta i valori dei vari parametri di riferimento come response time, utilizzazione sui dataitems, etc… influenzando in maniera sostanziale il comportamento del sistema.

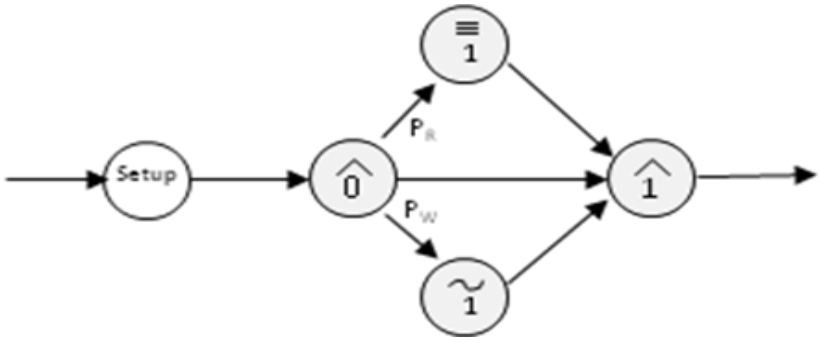
Il modello analitico è stato realizzato proprio per mettere in evidenza questa caratteristica e quindi non può essere considerato il più generale possibile in quanto rimane vincolato sotto le ipotesi che nella trattazione sono state effettuate al fine di semplificare la modellazione. Il modello esteso ha rilassato l'ipotesi estremamente vincolante della singola classe transazionale ma anch'esso è valido sotto altri vincoli che ne hanno reso più semplice la modellazione. Sulla base di questo quindi il modello potrebbe essere esteso maggiormente ed in particolare generalizzato per poter essere applicato in condizioni di carico generico. Le

ipotesi che sono tuttora presenti e che possono essere considerate maggiormente limitative introducendo dei forti paletti per l'applicazione del modello sono due:

- Considerazione di un'unica operazione di write e di conseguenza della presenza di soli lock esclusivi sui dataitems;
- Limitazione della coda di transazioni in attesa di un lock ad una soltanto;

La seconda ipotesi è stata introdotta per semplificare il lavoro ma non si considerava così stringente in quanto era già stata usata in molti dei lavori analoghi in letteratura. In realtà però, a valle della fase di validazione, si è potuto osservare come in presenza di una alta lock contention sui dati, l'ipotesi risultava troppo restringente e quindi non più applicabile. Questo potrebbe essere un motivo si sviluppo futuro per il modello, magari utilizzando elementi della teoria delle code per la modellazione delle transazioni in attesa sui locks.

Per quanto riguarda la prima ipotesi invece, il suo rilassamento prevede una modifica sostanziale al modello di transazione adottato in questo lavoro. Il modello, riportato in figura, dovrà prevedere non soltanto la possibilità di entrare in uno stato di attesa (tilde) a seguito di un'operazione di write bensì anche la possibilità di entrare in un altro stato di attesa dovuto alla contention generata da un'operazione di lettura come descritto anche nelle specifiche del protocollo.



Sommario delle formule

Modello base, unica classe transazionale

Response time delle transazioni	$R = R_{setup} + \hat{R}_0 + \sum_{k=1}^{M} \left(\hat{R} + \tilde{R}_k \right) + R_{commit}$
Utilizzazione dei set di dataitems	$Th_i = \sum_{k=0}^{M} \left[P_{i,k} * \left(\sum_{j=k}^{M} \hat{R} + \sum_{j=k+1}^{M} \tilde{R}_j \right) \right] + + R_{commit} * \sum_{k=0}^{M} P_{i,k}$
Probabilità di wait	$P_{Wi} = \lambda * Th_i$
Probabilità di contention	$\hat{P}_{CONTi,k} = \frac{\sum_{l=0}^{K} P_{i,l} * \hat{R}}{\sum_{k=0}^{M-1} \left[\sum_{l=0}^{K} P_{i,l} * \hat{R} \right] + \sum_{k=0}^{M-1} \left[\sum_{l=0}^{K} P_{i,l} * \tilde{R}_l \right] + \left[\sum_{k=0}^{M} P_{i,l} * R_{commit} \right]}$ $\tilde{P}_{CONTi,k} = \frac{\sum_{l=0}^{K} P_{i,l} * \tilde{R}_k}{\sum_{k=0}^{M-1} \left[\sum_{l=0}^{K} P_{i,l} * \hat{R} \right] + \sum_{k=0}^{M-1} \left[\sum_{l=0}^{K} P_{i,l} * \tilde{R}_l \right] + \left[\sum_{k=0}^{M} P_{i,l} * R_{commit} \right]}$ $P_{CONTi,Commit} = \frac{\sum_{l=0}^{M} P_{i,l} * R_{commit}}{\sum_{k=0}^{M-1} \left[\sum_{l=0}^{K} P_{i,l} * \hat{R} \right] + \sum_{k=0}^{M-1} \left[\sum_{l=0}^{K} P_{i,l} * \tilde{R}_l \right] + \left[\sum_{k=0}^{M} P_{i,l} * R_{commit} \right]}$

Tempi di attesa su i set di dataitems	$\tilde{R}_{Wi} = \sum_{k=0}^{M} \hat{P}_{CONTi,k} * \left(\frac{\hat{R}}{2} + \sum_{l=k+1}^{M} \hat{R} + \sum_{l=k+1}^{M} \tilde{R}_l + R_{commit} \right) + + \sum_{k=1}^{M} \tilde{P}_{CONTi,k} * \left(\frac{\tilde{R}_k}{2} + \sum_{l=k+1}^{M} \hat{R} + \sum_{l=k+1}^{M} \tilde{R}_l + R_{commit} \right) + + P_{CONTi,Commit} * \frac{\hat{R}}{2}$
Tempi di attesa sugli stati tilde	$\tilde{R}_k = \sum_{i=0}^{I} P_{i,k} * \tilde{R}_{Wi} * P_{Wi}$
carico di CPU	$\mathrm{C} = \mathrm{I}_{\mathrm{Setup}} + \sum_{k=0}^{M} \left[\left(P^R * I_{READ} \right) + \left(P^W * I_{WRITE} \right) \right] + \mathrm{I}_{\mathrm{Commit}}$
L'utilizzazione delle CPU	$\rho = \frac{\lambda \cdot \mathrm{C}}{\mathrm{k} \cdot \mathrm{MIPS}}$
Modello hardware	$\mathrm{p[attesa]} = \frac{\left(\frac{(\mathrm{k}\rho)^{\mathrm{K}}}{\mathrm{k!}} \cdot \frac{1}{1-\rho} \right)}{\left[\sum_{\mathrm{j}=0}^{\mathrm{k}-1} \frac{(\mathrm{k}\rho)^{\mathrm{j}}}{\mathrm{j!}} + \frac{(\mathrm{k}\rho)^{\mathrm{K}}}{\mathrm{k!}} \cdot \frac{1}{1-\rho} \right]}$ $\gamma = 1 + \frac{\mathrm{p[attesa]}}{\mathrm{k} \cdot (1-\rho)}$
Response time medio in uno stato di setup	$\mathrm{R}_{Setup} = \gamma \cdot \frac{\mathrm{I}_{Setup}}{\mathrm{MIPS}}$
Response time medio in uno stato di commit	$\mathrm{R}_{Commit} = \gamma \cdot \frac{\mathrm{I}_{Commit}}{\mathrm{MIPS}}$
Response time medio in uno stato cappello	$\hat{R} = \gamma \cdot \frac{\left(P^R * I_{READ} \right) + \left(P^W * I_{WRITE} \right)}{\mathrm{MIPS}} + \left[T_{IO} * \left(1 - P_{BH} \right) \right]$

Modello esteso, più classi transazionali

Response time di una classe transazionale	$R^c = \mathrm{R}_{\mathrm{setup}} + \hat{\mathrm{R}}_0 + \sum_{k=1}^{M_c} \left(\hat{\mathrm{R}} + \mathrm{R}_{\mathrm{k}}^{0/6} \right) + \mathrm{R}_{\mathrm{commit}}$
Response time medio	$R = \frac{\sum_{c=0}^{T} \lambda_c * R^c}{\sum_{c=0}^{T} \lambda_c}$
Utilizzazione dei set di dataitems di una classe transazionale	$\mathrm{Th}_{\mathrm{i}}^{c} = \sum_{k=0}^{M_c} \left[\mathrm{P}_{i,k}^{c} * \left(\sum_{\mathrm{j}=k}^{M_c} \hat{\mathrm{R}} + \sum_{\mathrm{j}=k+1}^{M_c} \tilde{R}_{\mathrm{j}} \right) \right] + \mathrm{R}_{\mathrm{commit}} * \sum_{k=0}^{M_c} \mathrm{P}_{i,k}^{c}$
Probabilità di wait	$P_{Wi} = \sum_{c=0}^{T} \lambda_c * \mathrm{Th}^{c}_{\mathrm{i}}$

Probabilità di contention per ogni classe transazionale	$$Den = \sum_{c=0}^{T} \left\{ \lambda^c * \left\{ \begin{array}{l} \left[\sum_{k=0}^{M_c-1} \left[\sum_{l=0}^{K} P^c{}_{i,l} * \hat{R} \right] + \right. \\ \left[\sum_{k=0}^{M_c-1} \left[\sum_{l=0}^{K} P^c{}_{i,l} * \tilde{R}^c{}_l \right] + \right. \\ \left[\sum_{k=0}^{M_c} P^c{}_{i,l} * R_{commit} \right] \end{array} \right\} \right\}$$ $$\hat{P}^c{}_{CONTi,k} = \frac{\lambda^c * \sum_{l=0}^{K} P^c{}_{i,l} * \hat{R}}{Den}$$ $$\tilde{P}^c{}_{CONTi,k} = \frac{\lambda^c * \sum_{l=0}^{K} P^c{}_{i,l} * \tilde{R}^c{}_k}{Den}$$ $$P_{CONTi,Commit} = \frac{\lambda^c * \sum_{l=0}^{K} P^c{}_{i,l} * R_{commit}}{Den}$$
Tempi di attesa su i set di dataitems	$$\tilde{R}_{Wi} = \sum_{c=0}^{T} \left[\begin{array}{l} \left[\sum_{k=0}^{M_c} \hat{P}^c{}_{CONTi,k} * \left(\frac{\hat{R}}{2} + \sum_{l=k+1}^{M_c} \hat{R} + \sum_{l=k+1}^{M_c} \tilde{R}^c{}_l + R_{commit} \right) + \right. \\ \left[\sum_{k=1}^{M_c} \tilde{P}^c{}_{CONTi,k} * \left(\frac{\tilde{R}_k}{2} + \sum_{l=k+1}^{M_c} \hat{R} + \sum_{l=k+1}^{M_c} \tilde{R}^c{}_l + R_{commit} \right) + \right. \\ P^c{}_{CONTi,Commit} * \frac{\hat{R}}{2} \end{array} \right]$$
Tempi di attesa sugli stati tilde di ogni classe transazionale	$$\tilde{R}^c{}_k = \sum_{i=1}^{I} P^c{}_{i,k} * \tilde{R}_{Wi} * P_{Wi}$$
Carico di CPU per classe transazionale	$$C^c = I_{Setup} + \sum_{k=0}^{M_c} \left[\left(P^R * I_{READ} \right) + \left(P^W * I_{WRITE} \right) \right] + I_{Commit}$$

Utilizzazione delle CPU	$\rho = \frac{\sum_{c=0}^{T} \lambda_c \cdot C^c}{k \cdot MIPS}$

Bibliografia

[1] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *"Concurrency Control and Recovery in Database Systems"*, Addison-Wesley, 1987.

[2] R. Agrawal, Michael J. Carey and M. Livny, *"Concurrency Control Performance Modeling: Alternatives and Implications"*. ACM Transactions on Database Systems, Volume 12 , Issue 4 (December 1987).

[3] J. F. Meyer, *"On Evaluating the Performability of Degradable Computing Systems"*, IEEE Transactions on Computers, Volume 29 , Issue 8 (Agosto 1980) .

[4] S. Jorwekar, A. Fekete, K. Ramamritham, S. Sudarshan, *"Automating the Detection of Snapshot Isolation Anomalies"*, VLDB '07, Settembre 2007.

[5] P.O'Neil, E. O'Neil, D.Shasha, A.Fekete. Progetto *"Isolation Testing for Transactional Systems"*.

[6] Transaction Processing Performance Council (TPC), *"TPC BENCHMARK™ W Specification"*, sito web: www.tpc.org.

[7] Wayne D. Smith, *" TPC-W: Benchmarking An Ecommerce Solution"*

[8] P. S. Yu, D.M. Dias, S.S. Lavenberg, *"On the Analytical Modeling of Database Concurrency Control"*, Journal of ACM, Vol. 40, No. 4, Sept. 1993, pp. 831-872.

[9] I.K.Ryu, A.Thomasian, *"Analysis of database performance with dynamic locking"*, Journal of he ACM (JACM), Volume 37 , Issue 3 (July 1990), pp. 491 – 523.

[10] B.Ciciani. D.M.Dias, P.S.Yu, *"Analysis of Concurrency-Coherency Control Protocols for Distributed Transaction Processing Systems with Regional Locality***"**, IEEE Trans. Software Eng., Vol. 18, No. 10, Oct. 1992, pp. 899-914.

[11] B.Ciciani. D.M.Dias, P.S.Yu, *"Dynamic and static Load sharing in Hybrid Distributed-Centralized Systems"*, Computer Systems Science and Engineering, Vol. 7, No. 1, Jan. 1992, pp. 25-41.

[12] http://www.tpc.org

[13] Transaction Processing Performance Council (TPC), "*TPC BENCHMARKTM App (Application Server) Specification, Version 1.1.2*", Dec 07, 2005, http://www.tpc.org

[14] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, Marta Simeoni, "*Software Performance: state of the art and Perspectives*", January 8, 2003.

[15] D. A. Menascè, V. A. F. Almeida, "*Capacity planning for Web performance. Metrics, Models and Method*", Prentice Hall, 1998

[16] L. Kleinrock, "*Queuing Systems Vol. 1: Theory*", Wiley-Interscience, 1975.

[17] J. Gray, P. Homan, R. Obermarck, H. Korth, "*A straw man analysis of probability of waiting and deadlock*", IBM Res. Rep. RJ 3066, San Jose, CA, 1981.

[18] S. Wu, B. Kemme, "Postgres-R(SI): *Combining Replica Control with Concurrency Control Based on Snapshot Isolation*", 21st International Conference on Data Engineering (ICDE'05) , pp. 422-433.

[19] A. Thomasian, I.K. Ryu, "*Performance analysis of two-phase locking*", IEEE Transactions on Software Engineering , Volume 17 , Issue 5 (May 1991) Pages: 386 – 402

[20] M. J. Carey, M. R. Stonebraker, "*The performance of concurrency control algorithms for database management systems*", Proceedings of the Tenth Intematlonal Conterence on Very Large Data Baeee., Singapore, August, 1984,

[21] R. Balter, P. Berard, P. Decitre, "*Why control of the concurrency level in distributed systems is more fundamental than deadlock management* ", Proceedings of the first ACM SIGACT-SIGOPS symposium on Principles of distributed computing, Ottawa, Canada, 1982.

[22] Y. C. Tay, R. Suri, N. Goodman, "*A Mean Value Performance Model for Locking in Databases: The No-Waiting Case*", Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems, Waterloo, Ontario, Canada, 1984.

[23] Transaction Processing Performance Council (TPC), "*TPC BENCHMARKTM C Specification, Revision 5.9*", June 2007

[24] A. Thomasian, "*Concurrency Control: Methods, Performance, and Analysis*", ACM Computing Surveys, Vol. 30, No. 1, March 1998.

[25] G. S. Fishman, "*Discrete-Event Simulation: Modeling, Programming, and Analysis*", Springer-Verlag, 2001.

[26] N.B. Al-Jumaha, H.S. Hassaneinb, M. El-Sharkawia, "*Implementation and modeling of two-phase locking concurrency"*", Information and Software Technology, Elsevier Science, 2000.

[27] N. Goodman ,HR. Sury, Y. C. Tray, "*A simple analytic model for performance of exclusive locking in database systems"*, Symposium on Principles of Database Systems, ACM, 1983.

[28] W. T.K. Lin, J.Nolte, "*Performance of Two Phase Locking*", Berkeley Workshop 1982

[29] D. R. Ries, Michael Stonebraker, **"***Effects of locking granularity in a database management system"***,** ACM Transactions on Database Systems (TODS), Volume 2 , Issue 3 (September 1977), ACM .

[30] D. R. Ries, Michael Stonebraker, *"Locking granularity revisited",* ACM Transactions on Database Systems (TODS), Volume 4 , Issue 2 (June 1979), ACM .