UNIVERSITA' DEGLI STUDI DI NAPOLI

"FEDERICO II"

FACOLTA' DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA ELETTRONICA

TESI DI LAUREA IN SISTEMI INFORMATIVI

Implementazione di uno scheduler transazionale a supporto di "mobile application"

Relatori

Ch.mo Prof. Antonio Picariello
Ch.mo Prof. Antonio D'Acierno
Ing. Vincenzo Moscato

Candidato

Francesco Saverio Caccavale

matr. 045/004597

ANNO ACCADEMICO 2005/2006

Indice

Ai miei genitori e a tutti i miei amici

Che c'e' di piu' duro d'una pietra e di piu' molle dell'acqua? Eppure la molle acqua scava la dura pietra.(Ovidio)

CAPITOLO 1

Introduzione al lavoro di tesi

Il seguente lavoro di tesi si colloca nell'ambito di ricerca noto con il nome di "Mobile Transaction Management", ed ha avuto come obiettivo principale la progettazione ed implementazione di uno scheduler transazionale per la validazione sperimentale di un modello teorico ibrido, per la gestione della concorrenza delle transazioni in ambiente mobile.
Dello scheduler transazionale si sono misurate le performance rispetto alla altre tecniche presenti nella letteratura del settore.
Il modello alla base dello scheduler può essere visto come una rivisitazione del noto protocollo di "Two Phase Locking (2PL)", che cerca di superare quelli che sono i limiti degli approcci classici per il controllo della concorrenza che mal si adattano alle caratteristiche di tali ambienti, tenendo, nel contempo, in conto la natura e la semantica delle transazioni in gioco.

1 Principali problematiche affrontate ed obiettivi del lavoro di tesi

Le caratteristiche tipiche di un dato ambiente applicativo potrebbero rendere le tradizionali tecniche per la gestione delle transazioni non del tutto appropriate.
Tali tecniche tendono a garantire le proprietà acide delle transazioni ovvero:

1. Atomicità
2. Consistenza
3. Isolamento
4. Persistenza

Di tali proprietà l'isolamento, ovvero la gestione della concorrenza, è quello che ha suscitato maggior interesse nei ricercatori.

Approcci pessimistici per il controllo della concorrenza, tra i quali il 2PL è il più conosciuto e adottato, si dimostrano non sempre efficienti in ambito distribuito per alcune tipologie di transazioni ovvero le "Long Lived Transaction". La durata maggiore di queste transazioni rispetto a quelle tradizionali non è legata al numero di operazioni effettuate dalla stessa ma piuttosto è conseguenza delle caratteristiche dell'ambiente in cui si opera.
Uno scenario in cui tale fenomeno è particolarmente sentito è quello delle transazioni in ambiente mobile.

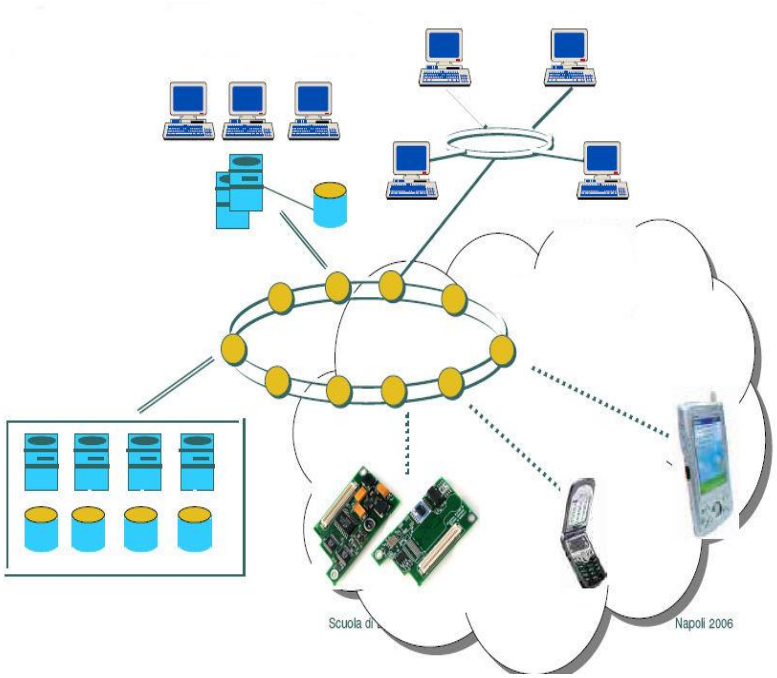


Figura 1 – Esempio di Ambiente Distribuito

Il rapido sviluppo delle tecnologie delle reti wireless e l'aumento delle capacità di calcolo dei dispositivi portatili hanno promosso un nuovo ambiente di lavoro mobile.
Un ambiente mobile è differente da un tradizionale ambiente distribuito a causa delle sue caratteristiche uniche:

1. mobilità degli utenti
2. frequenti e imprevedibili disconnessioni della rete wireless
3. limitazioni delle capacità di calcolo dei dispositivi mobili.

Un lungo periodo di disconnessione, dovuto anche a condizioni di hand-off, può causare l'inconsistenza dei dati o il blocco delle operazioni.
Per esempio, se il lock di un dato oggetto è detenuto da un dispositivo mobile disconnesso per un lungo periodo, ciò può ritardare l'esecuzione di applicazioni su altri hosts mobili.
La durata del periodo di disconnessione o connessione non è sempre pianificabile, variando nel tempo a causa di molti fattori, come ad esempio lo spostamento dell'utente mobile in un area in cui non è disponibile la rete wireless o l'esaurimento della batteria.
Disconnessioni imprevedibili potrebbero anche portare a periodi indefiniti di blocco delle risorse, che, per giunta potrebbero rivelarsi inutili in quanto non si ha la certezza che un utente mobile disconnesso abbia la possibilità o la voglia di riconnettersi per portar a termine la transazione che aveva iniziato.
Approcci ottimistici, alternativi a quelli precedentemente esposti, potrebbero sembrare più indicati per il contesto mobile.
Tali approcci si basano sull'assunzione che i conflitti sono rari, e quindi risulta più efficiente permettere alle transazioni di operare senza imporre ritardi per assicurarne la serializzabilità.
Tali tecniche potenzialmente permettono di incrementare di molto la concorrenza rispetto ai protocolli tradizionali perché non sono necessari blocchi delle risorse.
Quando una transazione vuole eseguire il commit si attua un controllo per determinare se vi sono stati dei conflitti.
In caso affermativo la transazione deve effettuare il rollback, dispendioso dal punto di vista computazionale, e poi riavviata.
Quindi per adottare questo approccio la premessa deve essere che i conflitti siano poco frequenti in modo tale da ridurre la probabilità di rollback.
Se da un lato quindi l'uso di approcci pessimistici può rendere i dati indisponibili per un lungo periodo di tempo, dall'altro meccanismi ottimistici possono portare ad una significativa divergenza tra il server e le copie locali dei dati, rendendo la loro reintegrazione difficoltosa.
Obiettivo del lavoro di tesi è quello di testare l'incremento della concorrenza delle transazioni in ambiente mobile utilizzando un protocollo che può essere considerato un ibrido tra approccio pessimistico ed ottimistico.

Per la validazione sperimentale del modello proposto è stato quindi implementato uno "scheduler transazionale" di cui si sono misurate le performance rispetto alla altre tecniche presenti nella letteratura del settore.
In analogia a quello che avviene nella gestione delle risorse in un sistema operativo, lo scheduler o GTM deve essere visto come un gestore delle risorse, dove queste sono rappresentate dagli LDBS.
Esso deve offrire un set di funzionalità base che consenta da un lato la gestione della concorrenza delle transazioni, e dall'altro l'amministrazione dell'intero sistema.
Insieme a tale set di funzionalità base, è richiesta poi l'integrazione di un insieme di servizi di supporto per la consistenza dei dati del database. Particolare importanza assumono in tal senso i tool classici per garantire le proprietà di atomicità, consistenza, isolamento e durability.

In particolare tale modulo, da un punto di vista funzionale ,dovrà offrire una serie di primitive per la gestione dell'accesso a dati condivisi da parte di più transazioni contemporaneamente attive. Il GTM (Global Transaction Manager) dovrà permettere le seguenti operazioni:

- gestire la concorrenza delle transazioni basandosi sulla semantica delle stesse;
- gestire le condizioni di deadlock;
- gestire il problema della starvation legato alle classi di compatibilità definite per le operazioni delle transazioni;
- gestire la momentanea disconnessione del client;

L'architettura ricalca quella *thin client* proposta da vari autori in letteratura , e la computazione è spostata lato server. Il client si immagina costituito da un semplice browser per la connessione alla rete wirless.
Il prototipo sviluppato in tecnologie JAVA e ORACLE PL/SQL costituisce un middleware fortemente portabile e facilmente integrabile con qualsiasi tipologia di piattaforma DBMS.

2 Organizzazione del lavoro di tesi

Il lavoro di tesi è organizzato come segue:

1. Nel I capitolo sono illustrati gli obiettivi del lavoro di tesi e le principali problematiche affrontate.
2. Nel II capitolo è riportato lo stato dell'arte sulla problematica della gestione delle transazioni in ambienti mobili.
3. Nel III capitolo è presentato e discusso il modello formale proposto per la gestione della concorrenza in ambienti mobili.
4. Nel IV capitolo è riportata la descrizione dello scheduler transazionale implementato e i casi d'uso dell'applicazione utilizzata per la validazione del modello teorico.
5. Nel V capitolo è riportata la validazione sperimentale del modello e il confronto con altri approcci noti in letteratura.

CAPITOLO 2

Stato dell'arte sulla gestione delle transazioni mobili

In questo capitolo vengono introdotti i vari metodi di controllo della concorrenza e in particolare i limiti degli approcci pessimistici, come il 2PL, in ambiente mobile. Vengono quindi descritti gli approcci in cui l'esecuzione di una transazione avviene o meno totalmente su rete fissa. Inoltre in merito ai vari approcci viene anche discussa la garanzia delle rimanenti tre proprietà acid delle transazioni.

Si discute infine degli approcci ottimistici per il controllo della concorrenza e della gestione del movimento e della disconnessione.

1 Contesto delle transazioni mobili

In questa sezione si discute di alcune caratteristiche relative al contesto delle transazioni mobili. Si attua a tale proposito la suddivisione seguente:

- contesto architetturale;
- contesto esecutivo.

1.1 Contesto architetturale

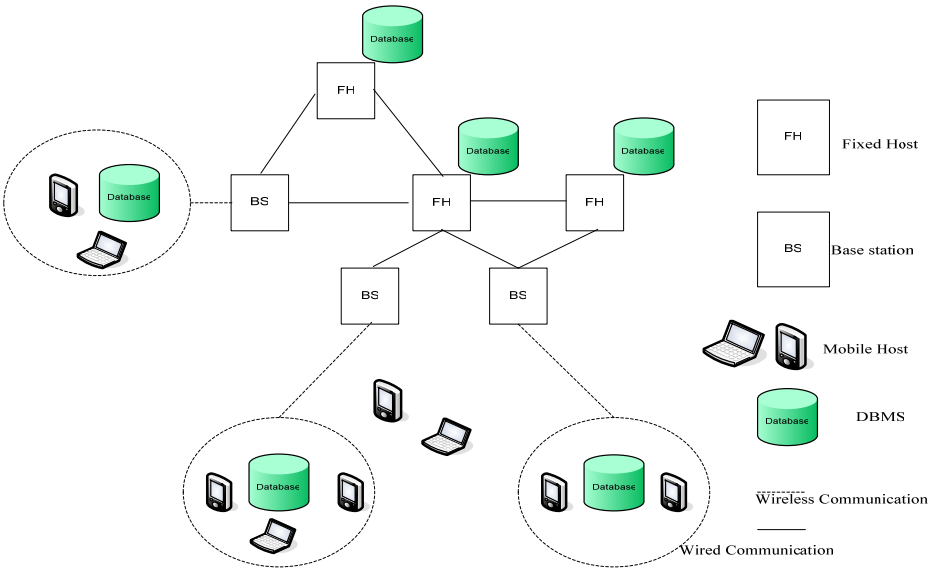


Figura 1 – Architettura globale dell'ambiente mobile

In una classica architettura client-server le funzioni di ogni attore sono definite staticamente [1].In assenza di fallimenti si assume che nè le localizzazioni del client e del server nè la connessione tra essi cambino.
In ambiente mobile, di contro, la distinzione tra client e server può essere temporaneamente nascosta risultando quest'ultimo un modello client server esteso [2].
La scelta architetturale impatta col progetto dell'applicazione e la gestione dei dati.
In particolare l'esecuzione delle transazioni su un MH è possibile solo se il MH possiede alcune capacità minime. I dati immagazzinati su un MH (memoria/disco) generalmente provengono dal data base server (FH) ed il lavoro del MH deve poi rimanere consistente con la base dati del FH.
Gli utenti mobili possono variare da clienti *thin* a clienti *full*, in funzione delle loro caratteristiche che seguono:

- **architettura thin client**: in tale tipologia di architettura l'elaborazione è eseguita completamente su server.
  Quest' architettura è appropriata per *dumb terminal* o piccole applicazioni di PDA. Le risorse del client sono limitate (piccola dimensione dello schermo, piccola memoria cache, limitata ampiezza di banda). Per quest'architettura il server è in carica per tutti i calcoli mentre i client possono osservare testo e grafica, ascoltare audio e video compresso, immagazzinare dati in ingresso;

- **architettura full client**: in ambiente mobile i client possono essere forzati a lavorare in modalità disconnessa o con connessioni deboli (sia per via della piccola ampiezza di banda, dell' alta latenza o degli alti costi). I *full client* emulano le funzioni del server per abilitare l' esecuzione delle applicazioni senza essere necessariamente connessi ai server remoti.
  I *full client* sono di solito pc portatili con risorse sufficienti per eseguire le applicazioni;

- **architettura client server flexible**: generalizza gli approcci *full client* e *thin client*. I ruoli dei client e dei server così come le funzionalità delle

applicazioni possono essere dinamicamente riallocati. La distinzione tra client e server può essere nascosta temporaneamente per fini prestazionali e di disponibilità dei dati.

- **architettura client-agent-server**: questo modello *three tier* introduce un agente o un proxy (procuratore) localizzato su rete fissa. Gli agenti sono utilizzati o in una varietà di ruoli come un surrogato di una o più MH oppure essendo attaccati ad uno specifico servizio o applicazione (esempio accesso al data base server).

1.2 Contesto esecutivo

Le transazioni mobili coinvolgono MH e FH. Come visto nella sezione 1.1, i server generalmente girano su FH (rete fissa) mentre i MH possono essere semplici client con alcune delle capacità del server. In accordo con le capacità del client possiamo definire cinque modelli di esecuzione.
I primi tre modelli coinvolgono un MH mentre il quarto e il quinto coinvolgono alcuni MH.

1.2.1 Completa esecuzione su rete fissa

In questo caso la MT (mobile transaction) è iniziata da un MH ma interamente eseguita sui FH. Questo è il classico approccio di *query shipping* dove il server esegue richieste di query di aggiornamento e invia il risultato al client. Esempi nel contesto mobile dove questo scenario è appropriato sono le query dipendenti dalla posizione (ad esempio hotel localizzati in un raggio di cinque miglia) e dagli aggiornamenti (voglio per esempio registrare una stanza in uno di questi hotel [3]).
In questo contesto è possibile eseguire transazioni su grossi insiemi di dati.

1.2.2 Completa esecuzione su un MH

In questo caso la MT è inizializzata ed eseguita sul MH. Questo modello richiede che il MH abbia tutti i dati rilevanti e abbastanza capacità computazionali per eseguire le sue transazioni locali.

L'autonomia dei MH consente di lavorare persino se alcune connessioni col server non sono disponibili. Procedure di riconciliazione sono necessarie per integrare il lavoro delle MH nel database server localizzato sui FH.

In molti casi alcune procedure finali vanno eseguite sul database server persino se la MT è eseguita sul MH. Per esempio consideriamo un viaggiatore che ha sul suo MH tutti i dati relativi ai prodotti che vende (prodotti disponibili, prezzi, etc.). Il lavoro fatto autonomamente (vendita) sul MH è integrato successivamente col server centrale.

1.2.3 Esecuzione distribuita tra MH e rete fissa

Questo modello è molto flessibile per permettere l'esecuzione di transazioni distribuite tra il MH e i database server su rete fissa. Questa distribuzione potrebbe essere motivata dalla disponibilità di risorse (dati, potenza del MH) o da ragioni di ottimizzazione. Le capacità dei server sono richieste sui MH per rendere minime le comunicazioni col server durante l'esecuzione delle transazioni. L' esempio delle vendite introdotto prima potrebbe anche richiedere un'esecuzione su uno scenario distribuito. Fino a che i venditori hanno le informazioni sui prodotti sui loro MH, potrebbero venderli senza connettersi al database server, warehouse store. Tuttavia la procedura di pagamento potrebbe richiedere una connessione al server della banca per controllare il credito del cliente. La vendita è fatta da una transazione distribuita essendo una sottotransazione eseguita su un MH e un'altra eseguita sul server della banca.

1.2.4 Esecuzione distribuita tra alcuni MHs

Questo caso è ambizioso ma interessante. L'obiettivo è prevedere un approccio peer to peer. I MH fanno da server per gli altri MH così l'esecuzione di una MT è distribuita tra alcuni MH.

L'idea è che in dipendenza dalla localizzazione dei MH potrebbe essere interessante chiedere al MH vicino di fare da server dati o da fornitore del servizio. Qui "vicino" è da intendersi in termini di comunicazione rispetto al database server.

Per esempio si considerino due uomini d'affari che lavorano nella stessa area geografica a cui serve condividere dei dati in modo cooperativo senza riferirsi al database server centrale. Per prevedere il modello di esecuzione particolari caratteristiche sono richieste per permettere ad ogni MH di essere consapevole di tutti gli altri.

Le BS potrebbero giocare un ruolo importante per mantenere cataloghi specifici del database permettendo ai MH di sapere i dati disponibili nell'area. Questi cataloghi potrebbero essere aggiornati e trasferiti sulle BS in accordo con i movimenti dei MH.

In reti ad-hoc [4,5], i MH interagiscono stabilendo una connessione punto a punto bypassando le BS.

Mentre nessuna BS è coinvolta ogni MH deve mantenere il suo catalogo del database e permettere ai vicini di accedervi. L'esecuzione di transazioni distribuite fra alcuni MH è particolarmente orientata alla configurazione di reti dinamiche.

1.2.5 Esecuzione distribuita tra alcuni MHs e FHs

Questo è lo scenario completamente distribuito dove l'esecuzione delle MT è distribuita tra alcuni host mobili e fissi.

L' approccio è un' estensione dello scenario precedente ed è orientato verso il lavoro cooperativo così come i multidatabase includono i MH partecipanti nell'esecuzione globale.

A tale scopo il commercio elettronico è un'applicazione promettente dove piccoli dispositivi potrebbero intraprendere transazioni commerciali tra esse stesse

(modello di esecuzione 4), con BS o con host remoti raggiunti attraverso una combinazione di infrastrutture senza fili o fisse [6].

Per esempio i MH partecipanti potrebbero essere in un ambiente aperto di commercio tipo fiera dove fornitori, produttori,commercianti e clienti che vorrebbero incontrarsi per vedere gli ultimi prodotti disponibili. I clienti potrebbero voler comprare alcuni prodotti dopo averli visti consultando dei cataloghi online disponibili alla fiera. Subito dopo la ricezione di un ordine il commerciante potrebbe contattarne un altro per evadere l'ordine.

Con dispositivi appropriati e supporto transazionale tutte queste operazioni potrebbero essere fatte su un sito e inviate più tardi ad un insieme di server.

I cinque modelli di esecuzione introdotti in questa sezione coprono tutte le possibilità che coinvolgono host fissi e mobili. Tuttavia i paragrafi successivi mostrano che le correnti proposte riguardano solo i primi tre modelli. La partecipazione di diversi host mobili nella stessa transazione distribuita non è stata ancora sviluppata.

2 Inadeguatezza delle proprietà acide per la gestione delle transazioni in ambienti mobili

Le proprietà acide (atomicità, consistenza, isolamento e durevolezza) si riferiscono alla correttezza delle transazioni.

La proprietà di atomicità è la nozione del tutto o niente: tutte le operazioni di una transazione devono essere eseguite o nessuna di esse. Il protocollo di commit assicura questa proprietà. La consistenza riguarda la correttezza del database. I database hanno di solito dei vincoli di integrità (per esempio restrizioni relative alle relazioni tra i dati) per mantenere il database in stati consistenti. Quindi ogni transazione deve trasformare il database da uno stato consistente ad un altro.

La proprietà di isolamento assicura che ogni transazione deve essere eseguita isolatamente anche se si pensa a transazioni eseguite in concorrenza. Protocolli per il controllo della concorrenza assicurano questa proprietà.

La durevolezza è la condizione per la quale gli effetti dei commit delle varie transazioni perdurino nel tempo. La registrazione di tutti gli eventi (logging) è una delle tecniche più usate per garantire la durevolezza.
Vediamo perché alcune regole classiche per garantire queste proprietà in ambiente mobile si rivelano inadatte.

2.1 Protocolli di commit

Il commit delle transazioni in ambiente mobile è affetto da periodi impredicibili e indefiniti di disconnessione dei MH.
Se una transazione richiesta da un MH client è completamente eseguita su rete fissa (modello di esecuzione 1) l'esecuzione del commit è veramente semplice. Una transazione può fare commit anche quando il MH è disconnesso, e messaggi pendenti o la ricezione dei risultati potrebbe essere differito alla successiva riconnessione. Consideriamo ora una transazione mobile eseguita su un MH che utilizza dati che provengono da un server su rete fissa (modello 2). In questo caso aggiornamenti fatti su un MH devono essere integrati e committati sul server. Un commit immediato sul server potrebbe non essere possibile a causa delle disconnessioni. Commit differiti sul server saranno necessari sebbene ciò potrebbe far crescere l'indice delle transazioni abortite.
Il protocollo di commit delle transazioni distribuite (modelli 3, 4 e 5) dovrebbe essere rivisti.
Il protocollo di 2PC standard utilizzato per le transazioni distribuite richiede un elevato numero di messaggi e non consente elaborazione non in linea. Inoltre è un protocollo bloccante perchè i partecipanti attendono per una decisione globale. Queste caratteristiche associate alle limitate risorse dei MH, disconnessioni e costi di comunicazione, rendono il 2PC non adeguato all'ambiente mobile.

2.2 Meccanismi di controllo della concorrenza

Le frequenti disconnessioni e la limitatezza delle comunicazioni influenzano anche il controllo dell'isolamento. Approcci pessimistici come il 2PL richiedono uno scambio di messaggi col server che potrebbero non essere sempre possibili. Disconnessioni imprevedibili potrebbero anche portare a periodi indefiniti di blocco delle risorse. Quindi approcci ottimistici sembrano molto più indicati per il contesto mobile.

2.3 Logging

Strategie che tollerino guasti in ambiente mobile sono particolarmente importanti.
In modalità disconnessa la "durevolezza locale" è la base per assicurare la "durevolezza globale" dei database server.
Le informazioni concernenti le elaborazioni locali vengono memorizzate nei log che possono essere utilizzati nei processi di riconciliazione per ottenere il commit finale sul database server.
Tuttavia per essere applicate in ambiente mobile le soluzioni tradizionali devono essere adattate. Ottimizzazioni per ridurre le dimensioni dei log e i tempi di elaborazione, divengono cruciali data la limitazione delle risorse dei MH. Inoltre per via della vulnerabilità dei MH ,come smarrimento o rottura, devono essere proposte strategie per la memorizzazione dei log su memoria stabile su rete fissa.
Tipicamente periodici trasferimenti dei log dai MHs alla BS corrente o FHs potrebbero essere abilitati.

2.4 Replicazione

Sebbene la replicazione dei dati non è necessariamente un problema transazionale, tale problema è stato affrontato da alcuni lavori sulle transazioni mobili.
La ragione è che i comuni approcci per migliorare l'autonomia dei MH sono basati sulla replicazione o memorizzazione dei dati.
Nei database tradizionali i requisiti di coerenza spesso portano a protocolli di replicazione che implementano un modello *one-copy equivalence*.

In questi modelli le repliche sono sempre equivalenti,la lettura da una replica fornisce sempre dati aggiornati. Negli ambienti mobili, i protocolli che lo sostengono (ad es. *ROWA* e *quorum-based*), devono occuparsi dei vincoli di risorse e comunicazione. Alcuni vincoli motivano l'uso di differenti tipi di repliche (una replica dovrebbe essere più piccola dell'oggetto originale) e i protocolli rilassano il modello di coerenza (esempio *lazy protocols* [7]) Questi protocolli potrebbero offrire repliche convergenti ma che possono essere lette o persino scritte su repliche divergenti, e in questi casi delle strategie di riconciliazione sono richieste.

In conclusione i requisiti di comunicazione potrebbero essere ridotti e i protocolli potrebbero tollerare disconnessioni.

E' anche importante notare che alcuni contributi di ricerca sulle transazioni mobili sono motivati da applicazioni dove agli utenti mobili possono richiedere meno "garanzie" sui dati rispetto ad una rete fissa. In questo caso le applicazioni lavorano con avanzati modelli transazionali, che rilassano alcune delle proprietà standard per provvedere a un compromesso tra consumo di risorse e qualità del servizio in un ambiente vincolato.

3 Modelli per la gestione delle transazioni in ambiente mobile

In questo paragrafo verranno presentati i vari approcci presenti in letteratura per la gestione delle transazioni in ambiente mobile.

Tali approcci possono essere suddivisi in:

- Approcci in cui l'esecuzione di una transazione avviene totalmente su rete fissa;
- Approcci in cui l'esecuzione di una transazione non avviene totalmente su rete fissa.

3.1 Approcci in cui l'esecuzione di una transazione non avviene totalmente su rete fissa

In questo paragrafo verranno spiegati i vari approcci per l'esecuzione di una transazione in ambiente mobile nel caso in cui l'elaborazione non avviene completamente su rete fissa e quindi lato client non basta un semplice browser.

3.1.1 Clustering

Clustering propone uno schema di replicazione per gli ambienti mobili dove i MH possono essere soggetti a varie disconnessioni. Esso assume un sistema pienamente distribuito ed è progettato per garantire la consistenza del database. La base di dati è divisa dinamicamente in *clusters*, ognuno raggruppa dati legati semanticamente o in base alla localizzazione. Un *cluster* potrebbe essere distribuito su vari host connessi fortemente. Quando un MH è disconnesso diviene esso stesso un *cluster*. Per ogni oggetto sono mantenute due copie, una di esse (versione *strict*) deve essere globalmente consistente, e l' altra (versione *weak*) può tollerare alcuni gradi di inconsistenza globale ma deve essere localmente consistente. Il grado di inconsistenza è una limitata divergenza tra la versione *strict* e *weak*. Quest'inconsistenza potrebbe variare in dipendenza della disponibilità dell'ampiezza di banda della rete tra i *cluster*.

Le MT possono essere sia *strict* che *weak*. Le transazioni deboli accedono solo a versioni deboli mentre le strette accedono a versioni strette .

Le transazioni strette sono eseguite quando gli host sono fortemente connessi e invece sono eseguite transazioni deboli quando i MHs sono debolmente connessi o disconnessi.

Due tipi di operazioni sono introdotte: *letture deboli* e *scritture deboli*. Transazioni strette contengono letture e scritture standard (operazioni strette), mentre transazioni deboli contengono operazioni deboli. Alla riconnessione un processo di sincronizzazione (eseguito sul database server) porta il database in uno stato di consistenza globale.

Le transazioni distribuite possono essere eseguite solo all'interno di un *cluster* come transazioni strette. I MH potrebbero partecipare ma solo in modalità connessa. In modalità disconnessa i MH eseguono solo transazioni deboli.

3.1.2 Two-tier replication

Questo metodo è nato dall'analisi condotta sugli schemi di replicazione *eager* e *lazy* [8]. Le analisi concludono che schemi forti non sono una buona opzione per ambienti mobili principalmente perchè non è possibile permettere le disconnessioni dei MH. Il metodo consiste in un meccanismo di replicazione debole che considera sia transazioni che approcci di replicazione per ambienti mobili dove i MH si connettono occasionalmente.
Esistono una versione principale per ogni dato e alcune versioni replicate (copie). Due tipi di transazioni sono supportate: *based transaction e tentative transaction.*
Le prime accedono alla versione principale (schema di replicazione *eager*) mentre le transazioni di tentativo accedono alla versioni di tentativo (schema di replicazione *lazy*).
Le transazioni di tentativo possono aggiornare i dati sui MH in modalità disconnessa.
Quando la connessione è stabilita le transazioni di tentativo sono rieseguite come transazioni base (coordinate dalla corrente BS) per raggiungere la consistenza globale. I risultati di questa riesecuzione potrebbero aver definito dei *criteri di accettazione* che permettono ai risultati di essere differenti. La riesecuzione delle transazioni permette agli aggiornamenti locali di persistere.

3.1.3 HiCoMo

HiCoMo [8] (High Commit Mobile) è un modello di transazione mobile dedito alle applicazioni decisionali.
L'obiettivo è permettere aggiornamenti durante le disconnessioni su dati aggregati memorizzati sui MHs. Esistono *tabelle base* su FHs dai quali si ottengono *tabelle aggregate*. Esse rappresentano una sintesi o statistiche (medie,somme,minimo,massimo) memorizzate sui MHs.
In modo simile al *clustering* e a *two tier-replication* si considerano due tipi di transazioni: *HiCoMo* e *transazioni base.*
Le prime sono eseguite su tabelle aggregate durante la disconnessione dei MHs. Le transazioni base riflettono le modifiche fatte dalle transazioni HiCoMo sulle

tabelle base. Quindi alla riconnessione una transazione HiCoMo è trasformata in più transazioni base -una per ogni tabella base alle quali si accede durante la generazione delle tabelle aggregate.

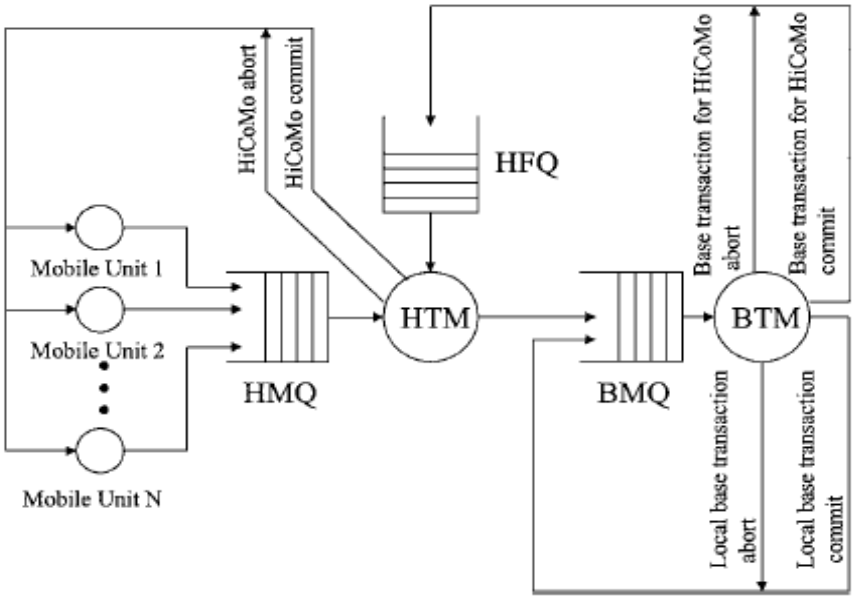


Figura 2 – Gestione delle transazioni mobili HICOMO

La trasformazione è basata su analisi complicate che fanno uso della semantica. Quindi il processo di trasformazione è il problema chiave del lavoro. Per ottenere un alta percentuale di successo di esecuzione vengono considerate solo operazioni commutative,addizioni e sottrazioni,per le transazioni HiCoMo e un margine di errore è tollerato tra transazioni HiCoMo e transazioni base rieseguite.

3.1.4 IOT

In IOT (Isolation – Only Transaction) le transazioni sono viste come sequenze di operazioni di accesso a file.

Le transazioni sono classificate in due categorie (similmente agli altri approcci):

- *prima classe* la cui esecuzione non contiene alcun accesso partizionato a file (cioè il client mantiene la connessione la server per ciascun accesso a file)
- *seconda classe* la cui esecuzione avviene nello stato disconnesso

Le transazioni di prima classe fanno il commit immediatamente dopo essere state eseguite, mentre la seconda classe si porta in uno stato pendente e attende per la validazione. Quando la riconnessione diviene possibile la seconda classe di transazioni è validata in accordo ai criteri di consistenza. (serializzabilità locale, serializzabilità globale, certificazione globale). Se la validazione ha successo i risultati sono integrati e poi viene eseguito il commit. Altrimenti le transazioni entrano nello stato di risoluzione. Questa potrebbe essere automatica (applicazioni specifihe di riesecuzione, abort) o manuale (notifiche degli utenti).

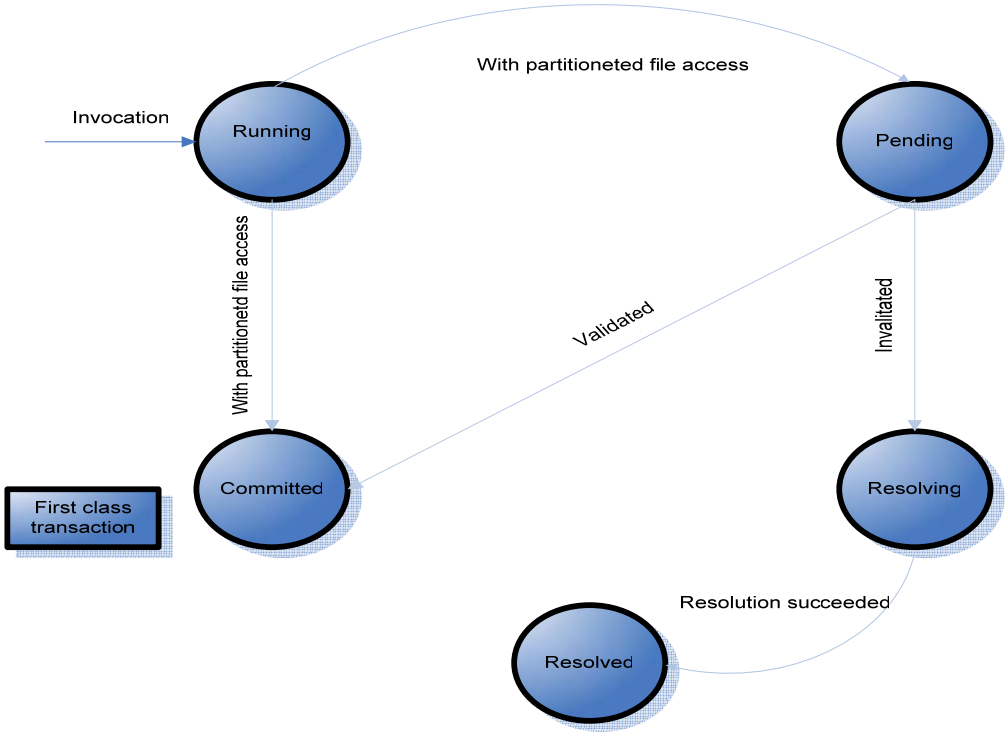


Figura 3 – Transizioni di stato in IOT

3.1.5 Pro-Motion

Pro-motion [9,10] si focalizza sul problema della memorizzazione dei dati (data caching) sui MH per rendere possibile l'elaborazione delle transazioni in modo consistente.

Pro-motion è un sistema di esecuzione di transazioni mobili che supporta la modalità disconnessa.

Per permettere l'esecuzione locale, sono introdotti i *Compact,* unità base di caching e controllo.

La semantica degli oggetti è utilizzata per la costruzione dei compact per migliorare l'autonomia e aumentare la concorrenza. Un compact racchiude tutte le informazioni necessarie per gestire ciò. Pro-motion utilizza le transazioni nested-split. Si considera l'intero sistema mobile come una *large long-lived transaction* eseguita su server. Le risorse necessarie per creare i *compact* sono ottenute da questa transazione attraverso usuali operazioni su database. La costruzione dei compact è fatta dal *compact manager* e dal *database server*. L'amministrazione dei compact è fatta da un *compact manager*, un *compact agent* sul MH e un *mobility manager* sulla BS.

Il compact manager potrebbe agire come programma di interfaccia per il database server ed appare come un ordinario database client che esegue una *large long-lived transaction*.

Su ogni MH il compact agent è responsabile della gestione della memorizzazione come pure dell'elaborazione delle transazioni, del controllo della concorrenza, del logging e del recupero. Il mobility manager si occupa delle trasmissioni tra agenti.

Le transazioni dei MH sono eseguite localmente anche in modalità connessa.

Un processo di sincronizzazione è eseguito dai compact agent e dal compact manager alla riconnessione. Questo approccio controlla i compact modificati localmente dalle transazioni che hanno eseguito il commit localmente. Se i compact preservano la consistenza globale allora il commit globale è eseguito.

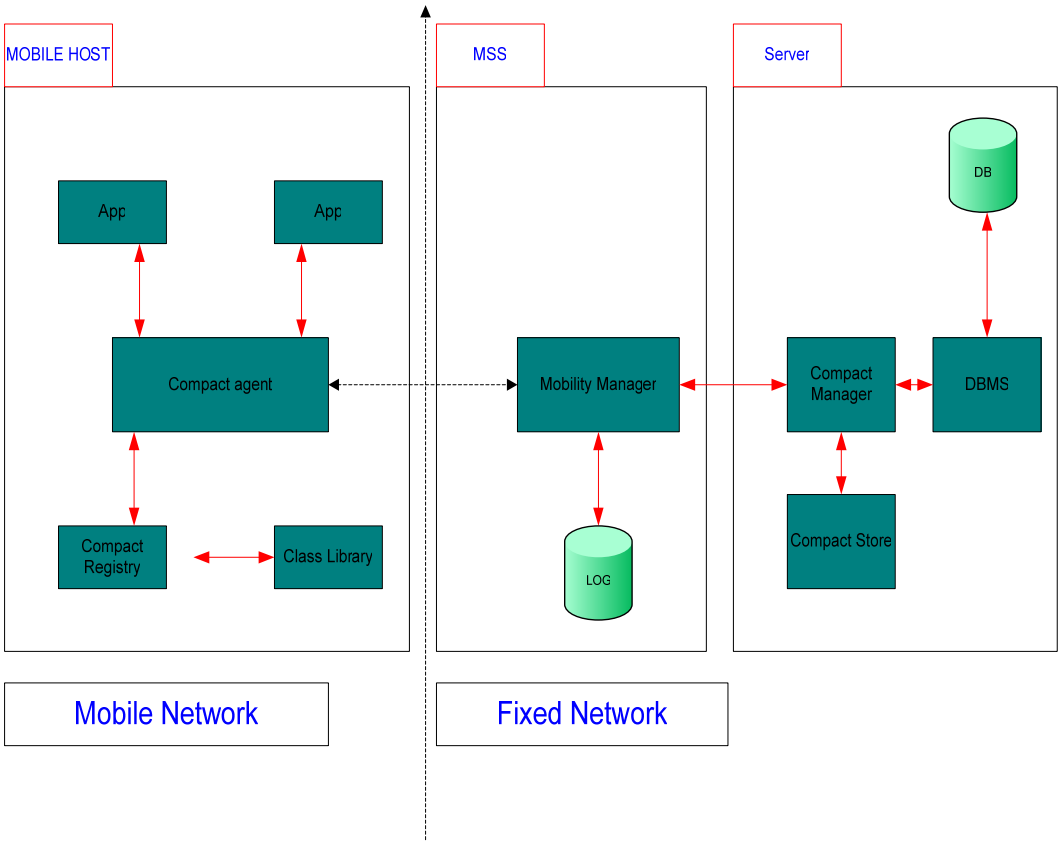


Figura 4 – Gestione delle transazioni mediante il modello Pro-Motion

3.1.6 Reporting

Reporting [11] considera l'ambiente del mobile database come uno specifico sistema multidatabase (MDBS) dove le transazioni su MH sono considerate come un insieme di sottotransazioni.

Reporting propone un modello di transazione open-nested che supporti:

Atomic Transaction; Non-compensatable Transaction; Reporting Transaction; Co-transaction.

Mentre sono in esecuzione le transazioni possono condividere i loro risultati parziali mantenendo lo stato della sottotransazione mobile (eseguita sul MH) su una BS.

Una transazione mobile è strutturata come un insieme di transazioni, alcune delle quali sono eseguite sul MH.

Gli autori considerano che la limitazione sui MHs rende necessario l'uso di un FH che memorizza o calcola per esempio parte della transazione.

Sono proposte *transazioni open-nested* con sottotransazioni del seguente tipo.

- *Atomic Transaction* hanno le proprietà standard per il commit e l'abort;
- *Non-compensatable Transaction* che al momento del commit delegano al padre tutte le operazioni invocate;
- *Reporting Transaction* che riportano ad un'altra transazione alcuni dei risultati ad un qualsiasi punto dell'esecuzione. L'annuncio potrebbe essere considerato come una delega dello stato tra transazioni.
- *Co-transactions* sono delle *reporting transaction* dove il controllo è passato dalle *reporting transaction* a quelle che ricevono il *report.* Co-transaction sono sospese all'istante della delega e riprendono l'esecuzione quando ricevono un report.

3.1.7 Semantic-based

Semantic-based si focalizza sull'uso di informazioni semantiche degli oggetti per migliorare l'autonomia dei MH in modalità disconnessa.
Questo metodo si concentra sulla frammentazione degli oggetti come soluzione per le operazioni correnti e per i limiti di memoria dei MH.
Questo approccio utilizza l'organizzazione degli oggetti e l'applicazione della semantica per dividere dati complessi in piccoli frammenti maneggevoli dello stesso tipo.
Ogni frammento può essere memorizzato indipendentemente e manipolato in modo asincrono. Le transazioni mobili sono invocate da un MH e dal punto di vista del server sono *long-lived transaction* per le possibili disconnessioni e il ritardo di comunicazione. Le richieste dei frammenti dei MH includono due parametri: criterio di selezione e condizioni di consistenza. Il criterio di selezione indica i dati da memorizzare sui MH e la dimensione richiesta dai frammenti. Le condizioni di consistenza specificano vincoli per preservare la consistenza dei dati interi. La frammentazione dei dati eseguita sul server permette un controllo della concorrenza ad elevata granularità. Copie principali dei frammenti sono date ai MH e le transazioni possono essere interamente eseguite su esse.
Un processo di riconciliazione è eseguito dal server quando si ha la riconnessione.
Questo modello potrebbe essere usato da differenti tipi di transazioni.

3.1.8 Prewrite

Prewrite [12] cerca di migliorare la disponibilità dei dati sui MH proponendo due varianti dei dati: *prewrite* e *write*. La variante prewrite riflette il futuro stato del dato ma potrebbe essere strutturalmente leggermente differente dal corrispondente valore scritto. I valori prewrite sono una versione minima dei valori scritti; perciò occorre una minore capacità di memoria sui MH. Per esempio in un oggetto di tipo documento il prewrite è il sommario e il write è il documento completo (sommario, corpo e bibliografia). In Prewrite, l'esecuzione della transazione è divisa tra MH e database server. Il *transaction manager* sui MH esegue le transazioni ma aggiornamenti permanenti sono fatti dal data manager localizzato sul server. Prewrite assicura che, delegando la responsabilità di scrivere al database server, l'elaborazione della transazione è ridotta sui MH. Tre operazioni – *prereads, prewrites e precommit* - da far eseguire al *transaction manager* sono proposte. Letture ordinarie e scritture permanenti sono fatte dal *data manager*. Le BS hanno capacità di fare il log delle operazioni e sono strettamente legate con il *data manager*. L'esecuzione delle transazioni procede come segue. Il *transaction manager* chiede alla BS la necessità di lock (in modalità connessa). La BS acquisisce i lock dal *data manager* e il MH può disconnettersi. Quando il transaction manager finisce la transazione, il commit locale (precommit) è fatto e riportato sulla BS. Il *data manager* rende le prewrite permanenti e fa il commit della transazione mobile. Prewrite considera le transazioni mobili come *long-lived transaction* che potrebbero essere realizzate con *nested* o *split transaction*.

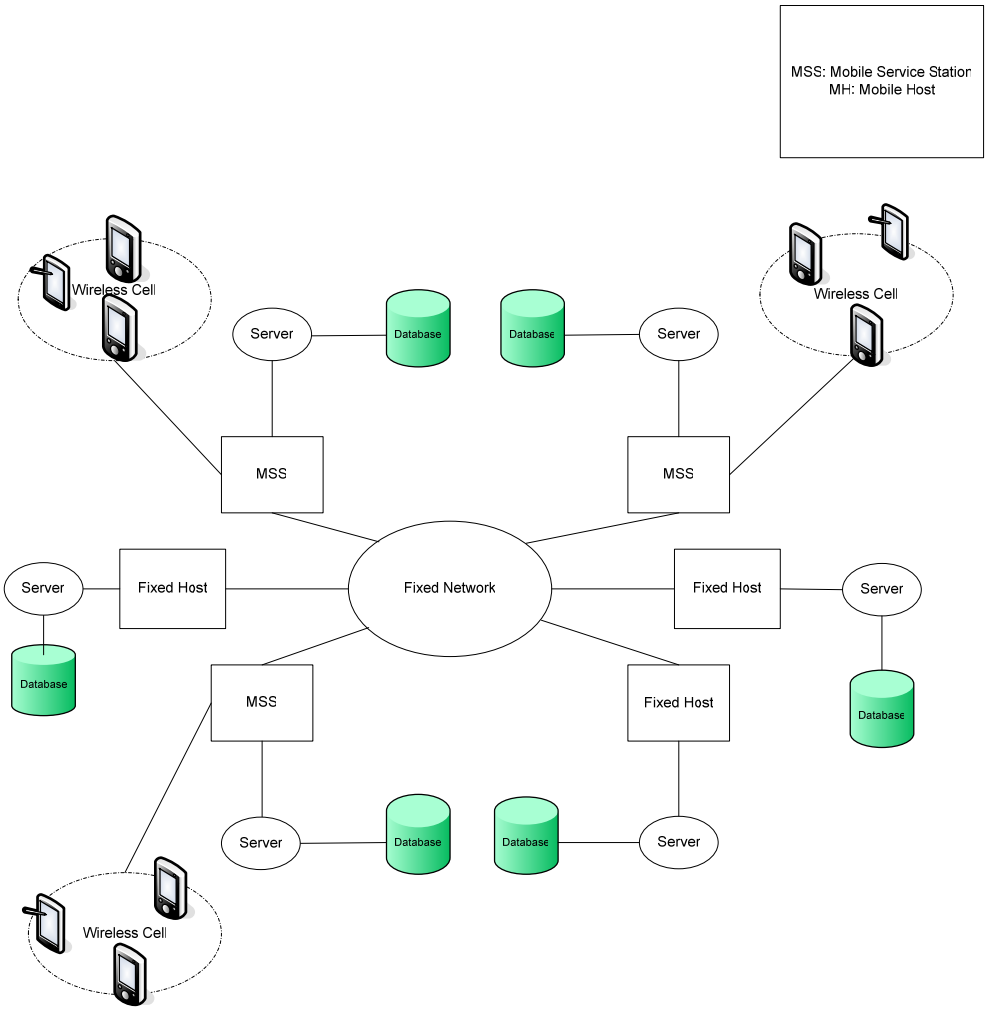


Figura 5 – Architettura nel caso del metodo Prewrite

Operation compatibility matrix.

	Pre-read	Read	Pre-write	Write
Pre-read	Yes	Yes	No	Yes
Read	Yes	Yes	No	No
Pre-write	No	Yes	No	Yes
Write	Yes	No	Yes	No

Figura 6 – Matrice delle operazioni compatibili nel metodo Prewrite

3.2 Approcci in cui l'esecuzione di una transazione avviene totalmente su rete fissa

3.2.1 Kangaroo Transaction

Kangaroo Transaction propone un modello di transazione mobile che si focalizza sul movimento di un MH durante l'esecuzione di una transazione. Le transazioni mobili sono transazioni globali generate da MH ed interamente eseguite su un *MultiDataBase System* (MDBS) su rete fissa.

KT propone di implementare un *Data Acces Agent* sopra un esistente *Global Transaction Manager*.

Questo agente è piazzato su tutte le BS e gestisce le Mobile Transaction e il movimento dei MH.

Ogni DBMS coinvolto ha la responsabilità di preservare le proprietà ACID delle sottotransazioni.

Il modello di transazione usa il concetto di transazioni:

- *Open-Nested*
- *Split*

L'esecuzione di una transazione mobile è coordinata dalla BS alla quale il MH è momentaneamente assegnato.

Quando un MH salta da una cella all'altra (conseguentemente da una BS all'altra) anche la coordinazione della transazione si muove.

Questa mobilità è catturata dividendo la transazione originale in due transazioni (chiamate Joey Transaction (JT)).

Lo split concerne solo la coordinazione della transazione.

In questo modo, se un MH salta da BS-1 a BS-2,BS-1 coordina le operazioni che sono eseguite durante la permanenza del MH nella cella di BS-1.

Le sottotransazioni sono eseguite sequenzialmente cosi tutte le sottotransazioni di JT-1 sono eseguite e fanno il commit prima di tutte le sottotransazioni di JT-2.

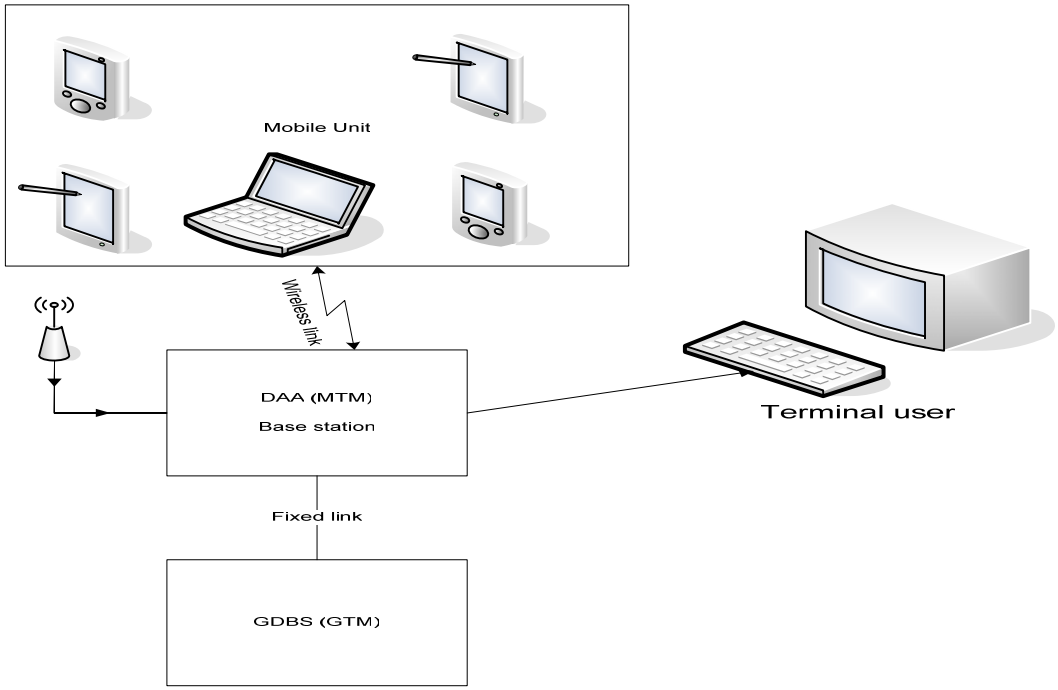


Figura 7 – Ambiente tipico delle Kangaroo Transaction

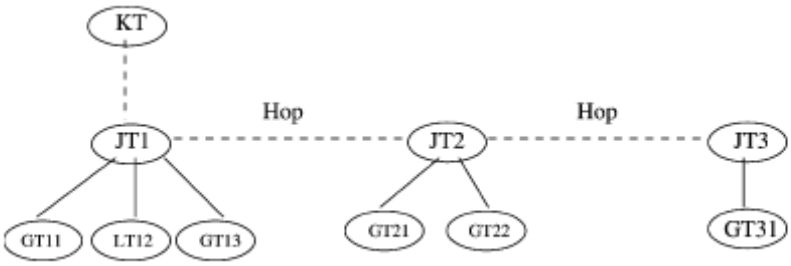


Figura 8 – Struttura gerarchica delle Kangaroo Transaction

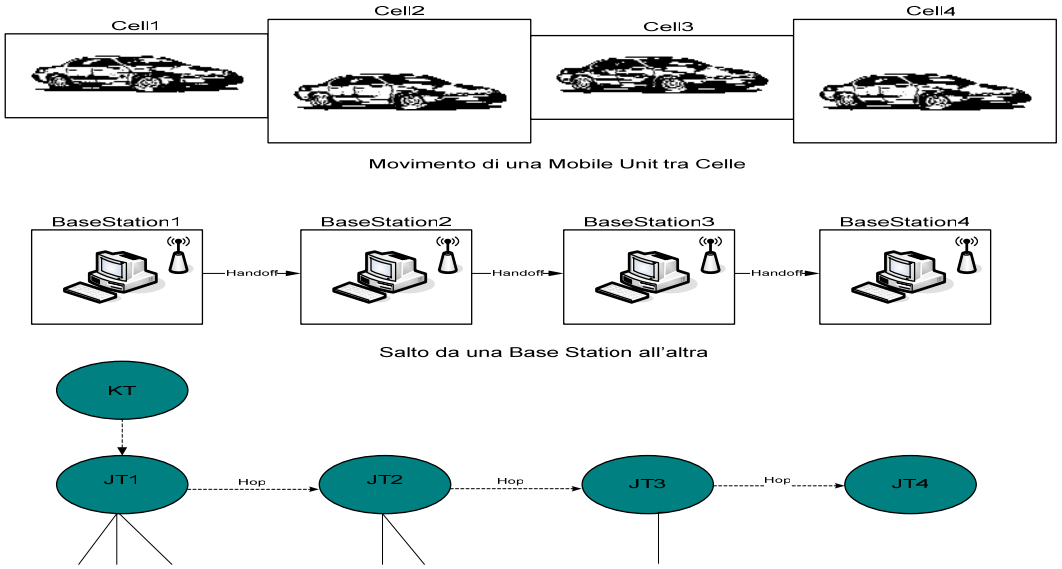


Figura 9 – Kangaroo Transaction

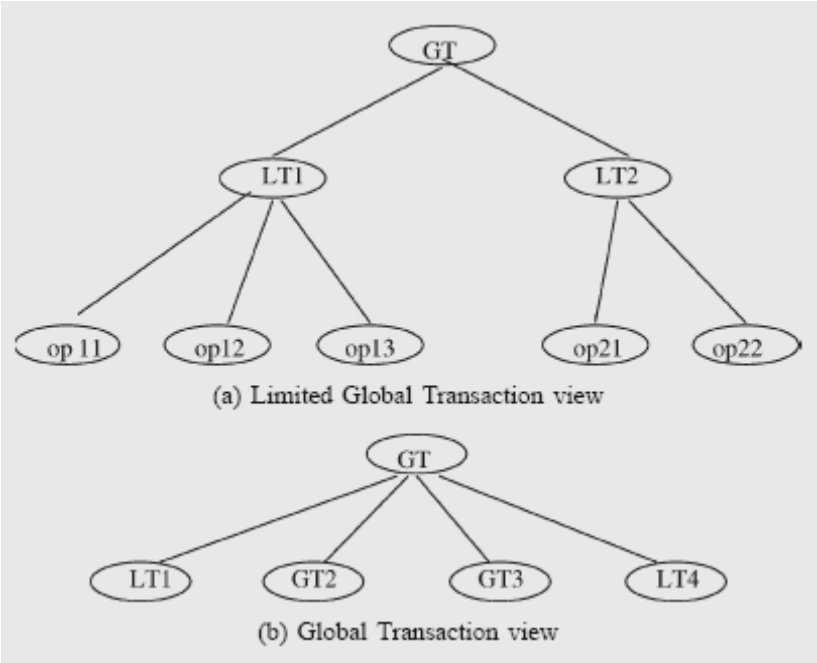


Figura 10 – Transazioni Globali

3.2.2 MDSTPM

MDSTPM [13] (Multidatabase Transaction Processing Manager) propone un framework di supporto per la richiesta di transazioni da parte di ogni MH in un ambiente multidatabase eterogeneo. Si ipotizza ci sia un MDSTPM su ogni host (FH) sull'esistente DBMS locale. I DBMS locali hanno la responsabilità delle elaborazioni locali. L' MDSTPM coordina l'esecuzione delle transazioni globali, genera lo scheduling e coordina i commit. Per via delle disconnessioni dei MH, un FH coordinatore è progettato in anticipo. Di conseguenza, una volta che un MH sottopone una transazione globale, potrebbe disconnettersi ed eseguire altri compiti senza dover attendere il commit della transazione mobile. L'host coordinatore gestirà la transazione mobile al posto del MH. In MDSTPM, come in KT, le principali proprietà acide vengono garantite da ogni DBMS su ogni sito della rete fissa.

3.2.3 Moflex

Il principale obiettivo di Moflex [14] è di consentire l'accesso a sistemi multidatabase eterogenei da parte dei MH.

Moflex supporta la gestione della mobilità e la flessibilità nella definizione e esecuzione delle MT. Estende il *Flexible Transaction Model* progettato per MDBS eterogenei dove una transazione è una collezione di sottotransazioni collegate da un insieme di dipendenze di esecuzione: successo, fallimento e dipendenze esterne (tempo, costo o localizzazione).

Accanto alle transazioni flessibili, Moflex permette la definizione di sottotransazioni dipendenti dalla localizzazione [3] e il supporto per l'adattabilità nell'esecuzione delle sottotransazioni quando si ha un hand-off.

Gli autori assumono che il sistema è costruito su eterogenei ed autonomi MDBS.

Gli eterogenei e mobili MDBS sono costituiti da tre strati: MH,BS ed MDBS.

Nello strato MH gli utenti definiscono le transazioni Moflex che sono sottoposte al *mobile transaction manager* della corrente cella wireless nello strato BS. Il *mobile transaction manager* coordina l'esecuzione delle transazioni sottoposte. Un *global transaction manager* sullo strato eterogeneo MDBS esegue transazioni garantendo le proprietà acide.

3.2.4 Pre-serialization

Pre-serialization [15,16] è orientata ad autonomi mobile MDBS. I MH richiedono l'esecuzione di transazioni ai MDBS dove ogni DBMS gira su FH. Le transazioni mobili sono considerate *long-lived transaction* composte da sottotransazioni compensabili (dette *site transaction*). Pre-serialization a differenza di KT,MDSTPM e Moflex ,fa rispettare l'atomicità e l'isolamento delle transazioni globali tenendo conto delle disconnessioni e migrazioni degli utenti mobili. Per minimizzare gli effetti delle transazioni mobili (*long lived transaction* a causa della disconnessione degli utenti) pre-serialization permette alle *site transaction* di eseguire il commit indipendentemente dalla transazione globale. Ciò permette il rilascio delle risorse locali in modo tempestivo. In più un processo di *pre-serialization* è portato avanti. Per questo un algoritmo basato su un grafo parziale di serializzabilità globale (PGSG) verifica la serializzabilità delle transazioni globali.

Il global transaction manager è costituito da due strati:

- Global Coordinator
- Site Manager

Lo strato globale consiste in un insieme di *global transaction coordinator* localizzati su ogni BS e su ogni altro nodo che supporta utenti esterni. Lo strato locale consiste in un insieme di site *transaction manager* su ogni DBMS partecipante. Transazioni globali sono richieste da utenti mobili grazie a un coordinatore globale che sottopone le *site transaction* ai manager locali.

Lo strato globale gestisce anche le disconnessioni, le migrazioni di utenti mobili, i log dei responsi che non possono essere inviati ad utenti non in linea e l'esecuzione dell'algoritmo PGSG.

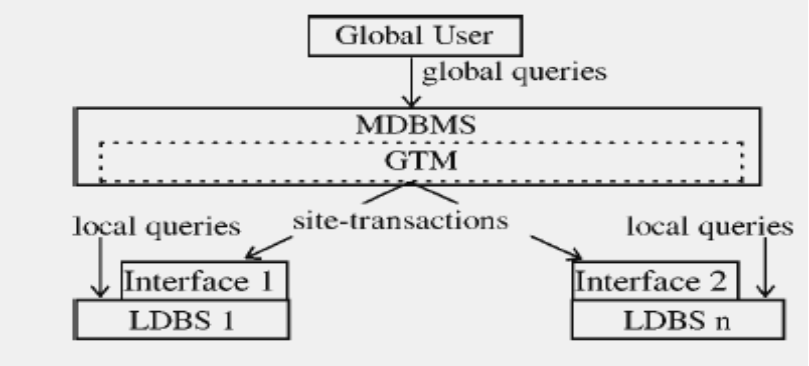


Figura 11 – Sistema mobile multidatabase

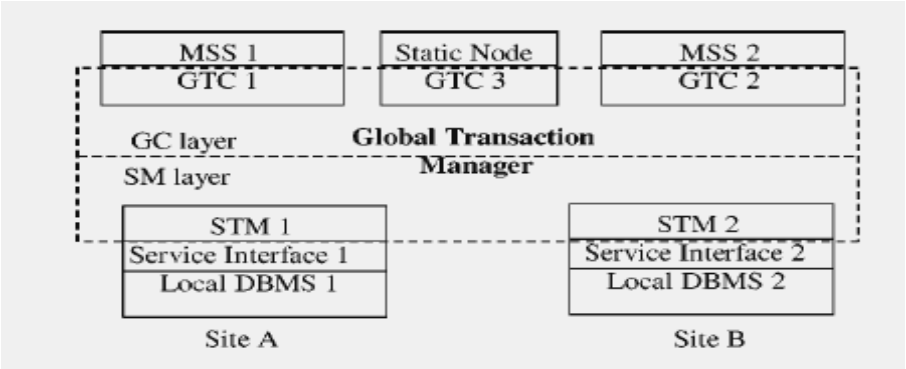


Figura 12 – Global Transaction Manager

4 Garanzia delle proprietà ACID per mobile transaction

In questo paragrafo verrà presentato come le MT gestiscono le proprietà ACID (*Atomicity, Consistency, Isolation, Durability*). Sono confrontati i lavori su esposti e identificate le caratteristiche comuni. L'analisi è stata divisa in base alle quattro proprietà. I lavori analizzati nella sezione sono quelli la cui esecuzione non avviene totalmente su rete fissa. Gli altri lavori verranno analizzati nel paragrafo seguente in quanto, ad eccezione di Pre-Serialization, si focalizzano maggiormente sul controllo della mobilità e non su un protocollo orientato alle proprietà ACID.

4.1 Atomicità

La proprietà di atomicità è assicurata dal protocollo di commit.

4.1.1 Protocollo di commit

Eccetto che per *Reporting*, il commit delle transazioni è eseguito in due fasi.
La prima è realizzata sui MH - commit locale - e la seconda - commit - dalla BS/Database server. Clustering, Two-tier replication, HiCoMo, IOT, Pro-motion and Prewrite eseguono il local commit, ognuno con caratteristiche specifiche:

- *Clustering* e *Two-tier replication* eseguono il commit locale solo in modo disconnesso utilizzando speciali tipi di transazioni. In modalità connessa è utilizzato un protocollo di commit atomico (per esempio 2PC) che include la partecipazione di alcuni host.

- *HiCoMo,Pro-motion* e *Prewrite* non differenziano le modalità connessa e disconnessa. Il commit locale è effettuato utilizzando un protocollo di commit atomico (2PC in Pro-motion).In Pro-motion il progettista della transazione decide se la transazione deve o meno fare il commit locale.

- *IOT* anche esegue un commit locale (nelle modalità connesse e disconnesse) ma il recupero in caso di fallimento non è garantito. Sui client mobili lo spazio è una risorsa limitata e occorre una grossa quantità di spazio per disfare l'effetto di una transazione. Conseguentemente il servizio di log potrebbe essere non disponibile al client quando lo spazio è esaurito. Le transazioni che hanno fatto il commit locale vanno in uno stato pendente.

Nella seconda fase del processo di commit, le transazioni che hanno eseguito il commit fanno aggiornamenti permanenti su database server. Il commit delle transazioni può coinvolgere meccanismi di riconciliazione o riesecuzione di transazioni.

- La riconciliazione in *Clustering* è fatta sintatticamente dove transazioni deboli sono abortite o rolled back se le loro scritture deboli vanno in conflitto con transazioni forti.

- Nel *Two-Tier Replication*, le transazioni base (riesecuzione delle *tentative transaction*) sono eseguite nell'ordine in cui sono state eseguite nel commit locale. Se questa riesecuzione fallisce, sebbene si tenga conto dei *criteri di accettazione* (legati ad ogni *tentative transaction*) le transazioni di tentativo sono abortite. Per migliorare le chances di successo, le transazioni di tentativo possono essere progettate per commutare insieme ad ogni altra.

- In *HiCoMo* il set di transazioni base generate da una HiCoMo è organizzato in un'estesa transazione innestata per la quale ogni transazione base è una sottotransazione. Se una transazione base è abortita per via dei vincoli di integrità un'altra transazione base può essere generata (dallo stesso HiCoMo) ed eseguita. Il criterio per fermare il nuovo processo dipende dalla definizione dei valori dei margini di errore. Quindi il global commit è quasi sempre garantito grazie alle considerazioni fatte. Le transazioni HiCoMo sono commutative, margini di errore sono tollerati e la riesecuzione di transazioni base è permessa.

- *IOT* prevede quattro opzioni per riconciliare transazioni pendenti:
  1) riesecuzione delle transazioni usando i file aggiornati sul server (questa è l'opzione di default);
  2) invocazione dell'applicazione specifica risolutrice della transazione (ASR), ovvero il progettista della transazione potrebbe unire un ASR ad una transazione per far si che venga automaticamente invocata dal sistema;
  3) abort della transazione;
  4) richiesta agli utenti di risolvere i conflitti manualmente.

- In *Pro-Motion* viene effettuato un controllo sui *compact* che hanno effettuato il commit localmente. Se alcuni compact non sono più validi,allora le transazioni mobili sono abortite e una *procedura contingente* (associata ad ogni commit locale) è eseguita per ottenere l'atomicità semantica.

- In *Prewrite*, nè la riconciliazione nè la riesecuzione sono eseguite.
  Grazie all' algoritmo di elaborazione delle transazioni e del protocollo di lock, *Prewrite* assicura che le transazioni che hanno eseguito il commit localmente (precommittate) lo eseguiranno sul database server. Ciò perchè le due varianti dei dati, *prewrite* e write, sono in realtà differenti.
  Perciò,le preread, prewrite ed i precommit eseguiti sui MH sono differenti dalle operazioni di read,write e commit eseguiti per rendere gli aggiornamenti permanenti (anche se tra esse vi è una particolare relazione).

- Il processo di commit è differente in *Reporting*, infatti ogni sottotransazione è atomica ma ciò non implica l'atomicità della transazione mobile globale. Le transazioni compensabili possono essere associate a sottotransazioni (in base alla semantica) quindi l'atomicità è garantita. *Reporting* nel caso di delega delle *co-transaction* non inficia l'atomicità perchè esse non richiedono alla *reporting transaction* un'operazione distinta per il commit e per l'abort. Una transazione è quasi atomica se tutte le operazioni per le quali essa è responsabile eseguono il commit o nessuna di esse. Le sottotransazioni potrebbero fare commit o abort unilateralmente senza aspettare ogni altra sottotransazione e persino le loro transazioni genitrici.
  - In *Semantic-based*, le transazioni sono considerate *long-lived transaction*. Visto che i MH sono responsabili del commit delle transazioni locali, potrebbe essere possibile supportare transazioni atomiche o meno. La tabella seguente sintetizza i processi di commit.

	Commit process	
	First step at MH	Second step at BS/DB server
Clustering	Disconnected mode: *local commit* of weak transactions. Connected mode: 2PC for strict transactions	*Commit* involves syntactic reconciliation with abortion and rollback in the resolution of conflicts
Two-tier replication	Disconnected mode: *local commit* of tentative transactions. Connected mode: atomic commit protocol for base transactions	Tentative transactions are re-executed taking into account their acceptance criteria
HiCoMo	*local commit* of HiCoMo transactions	Execution of base transactions taking into account defined margin errors. If a base transaction aborts, another one can be defined and executed
IOT	*local commit* of local transactions	Four resolution options for second class transactions: re-execution, application specific, abortion and notification to users
Pro-motion	*local commit* of local transactions (2PC)	A synchronization process checks compacts involved in local transactions. In case of conflicts, local transactions are aborted and contingency procedures are executed
Prewrite	*local commit* of local transactions (prewrite operations)	Local updates are made permanent by the write operations
Semantics-based	*local commit*	Updates reintegration (merge). As fragments are exclusive copies and they have attached consistency conditions there are no conflicts in reintegration.
Reporting	All subtransactions are atomic and they are able to commit independently of the parent transaction. In case of abortion compensating transactions can be associated to subtransactions (except for non-compensatable ones)	

Figura 13 – Sintesi dei vari protocolli di commit

4.1.2 Altri protocolli di commit

Si estendono le analisi considerando altri lavori dove i partecipanti potrebbero essere host mobili o fissi (modelli di esecuzione 3-5). La motivazione di questi protocolli generalmente è di provvedere a processi di commit mobili che tengano conto:

- delle limitate caratteristiche della rete wireless riducendo il numero di messaggi
- della natura mobile dei MH includendo BS nel processo di commit.

UCM (Unilateral Commit Protocol) [17] supporta disconnessioni ed esecuzioni non in linea (su MH). Questo lavoro è motivato dalla debolezza del protocollo di 2PC quando viene eseguito in ambienti mobili: elaborazione non in linea, necessità del MH di supportare lo stato di preparazione e il doppio invio di messaggi.
UCM è un protocollo ad un'unica fase dove la fase di votazione del 2PC è eliminata. Il coordinatore agisce da "dittatore" e diffonde le decisioni a tutti i partecipanti. Sono fatte alcune assunzioni. Per esempio a tutti i partecipanti è richiesto di serializzare le loro transazioni usando il 2PL stretto e al momento del commit gli effetti di tutte le transazioni locali dovrebbero essere loggati su memoria stabile. UCM garantisce atomicità e durevolezza; tuttavia per via di queste assunzioni, i dati ai quali hanno acceduto le transazioni locali non committate sono bloccati fino al commit e i log devono essere eliminati (sul FH) ad ogni commit della transazione.

TCOT (Transaction Commit On Timeout) [18,19] utilizza dei timeout per fornire un protocollo non bloccante con un numero ristretto di comunicazioni. Per raggiungere tale obiettivo invece di usare messaggi per sapere se un utente mobile è pronto per il commit, il coordinatore del commit aspetta per un certo timeout prima di morire. Il coordinatore è installato sulla corrente BS del MH, oppure salta da BS a BS insieme al MH. Quindi i partecipanti devono inviare un messaggio di commit al coordinatore. Se il timeout dei partecipanti scade e il coordinatore non ha ricevuto nè un messaggio di commit nè di abort, la transazione è abortita. Se è stato ricevuto il commit di tutti i partecipanti prima del timeout globale, la transazione fa il commit senza inviare un messaggio di commit globale. Se necessario i timeout potrebbero essere rinegoziati durante l'esecuzione.
I partecipanti potrebbero fare il commit localmente prima del commit globale. Se il commit globale fallisce il coordinatore invia un messaggio di abort globale e sono eseguite le transazioni di compensazione. Per essere capaci di fare il commit indipendentemente, i partecipanti mobili devono inviare i loro log di aggiornamento al coordinatore.
Quando il commit globale ha successo, il coordinatore invia gli aggiornamenti dei MH ai corrispondenti DBMS su rete fissa (i problemi di riconciliazione non sono

considerati). Il protocollo considera la modalità "*doze*" ma non tiene conto delle disconnessioni. TCOT assicura l'atomicità semantica [20]. Per provvedere all'atomicità (non atomicità semantica) e per evitare abort a catena, TCOT propone di usare il 2PL stretto per ogni partecipante.

4.1.3 Discussioni sui vari approcci

Semantics-based, *Pro-motion*, *Prewrite* e *Reporting* considerano le transazioni come *long-lived transaction*. Se queste transazioni sono eseguite su un MDBS, l'atomicità globale dipende dall'autonomia di ogni sistema di database [21].Se alcuni DBMS non possono partecipare ad un protocollo di commit atomico globale l'atomicità è difficile da garantire. Se TCOT e UCM venissero applicati a un MDBS, l'autonomia sarebbe violata perchè i DBMS sono vincolati a inviare i loro log al coordinatore del commit (in TCOT) o ad un FH (UCM).
Inoltre UCM e TCOT assumono che tutti i sistemi di elaborazione dei partecipanti utilizzano il 2PL stretto (questa assunzione può essere rilassata per TCOT).
Tali assunzioni potrebbero portare a bloccare i dati per periodi di tempo indefiniti. Tuttavia sono evitati abort a catena. Quest'ultimi potrebbero invece aversi in *Clustering*, *Two-tier replication* e *Pro-motion*.
Ma visto che il commit locale delle transazioni modifica dati locali, è generato solo l'abort delle transazioni locali. In più questi abort riguardano solo *weak transaction* e *tentative transaction* perchè i risultati locali sono disponibili esclusivamente per questo tipo di transazioni. IOT previene l'inconsistenza a cascata avvertendo gli utenti di quali sono gli oggetti ai quali hanno avuto accesso transazioni non ancora validate (detenute da transazioni non ancora validate).
Nella riconciliazione, quando una *non-validated transaction* è stata risolta, sia lo stato globale che locale dell'inconsistenza degli oggetti devono essere mostrati, in questo modo il *revolver* può scegliere lo stato locale o globale per un oggetto.
Riguardo ai lavori che utilizzano un modello di esecuzione distribuita (*Clustering* con *strict transaction*, *Two-Tier Replication* con *base transaction, UCM* e *TCOT*), solo *UCM* supporta la disconnessione dei MH.
In *Clustering* e *Two-Tier Replication* i MH dovrebbero essere saldamente connessi. Se avviene una disconnessione il processo di commit incorre in errori.

In TCOT, se un partecipante mobile fa il commit localmente e immediatamente dopo si disconnette, potrebbe non essere informato dell'abort globale successivo.
Il commit delle transazioni mobili è generalmente eseguito in due fasi: il commit locale è fatto su MH e il commit che è fatto sulla BS/Database server. Questo approccio rilassa l'atomicità e richiede l'esecuzione di qualche processo in più rispetto alle tecniche tradizionali (esempio 2PC).
Tuttavia esso si adatta agli ambienti mobili perchè fornisce ai MH la possibilità di lavorare in modalità disconnessa senza bloccare l'esecuzione del sistema.

4.2 Consistenza

La consistenza è garantita rispettando i vincoli di integrità che si basano sulle applicazioni - per definirli viene utilizzata l'informazione semantica delle applicazioni. Si analizzano i modi in cui l'informazione semantica è utilizzata per assicurare la consistenza dei dati quando si utilizzano le transazioni mobili. La sezione 4.2.2 sintetizza le analisi.

4.2.1 Informazione semantica

In *Clustering* l'informazione semantica è usata per specificare il grado di inconsistenza tra i *cluster*. Questo grado potrebbe essere limitato dal:
1) numero di commit locali
2) numero di transazioni che potrebbero operare su copie inconsistenti
3) numero di copie che possono divergere
Esiste anche una funzione h di proiezione che controlla questo grado proiettando operazioni *strict* sulle versioni dei dati *weak* . La piena consistenza è ottenuta riconciliando differenti copie degli stessi dati localizzati su *cluster* differenti.
In *Two-tier replication*, *acceptance criteria* è un test che permette ai risultati delle transazioni base di essere lievemente differenti dalle *tentative transaction*. Questa differenza accettabile è basata sulla semantica. Quest'ultima è anche utilizzata per progettare *tentative transaction* commutative.

In HiCoMo l'informazione semantica è usata:

- per ottenere le tabelle aggregate
- per progettare transazioni commutative HiCoMo
- per definire margini di errore permessi
- generare transazioni base.

In particolare, la funzione di trasformazione della transazione (usata per generare le *base transaction*) ha bisogno come ingresso:

- delle tabelle aggregate alle quali si accede
- dei tipi di operazione
- della configurazione delle tabelle base
- dei vincoli di integrità e dei conflitti tra transazioni base concorrenti e la relativa transazione HiCoMo.

In IOT, l' *Application specific resolver* (ASR) applicato alle transazioni pendenti è basato sull'informazione semantica.

Pro-motion e *Semantic-based* sfruttano l'informazione semantica per costruire i *compact* e i frammenti:

- Per *Pro-motion* il *compact* rappresenta un accordo tra il database server e il MH. Il *compact manager* e il database server incapsulano in *compact*: dati, tipi di metodi specifici, informazioni sullo stato, regole di consistenza e vincoli. Se l'agente dei compact e il *compact manager* rispettano tutte queste condizioni, l'uso dei *compact* potrebbe non influire sulla consistenza del database. Il progetto dei *compact* potrebbe richiedere criteri di correttezza e metodi per il controllo della concorrenza per *compact*.

- In *Semantic-based*, per preservare la consistenza, gli oggetti devono supportare divisione e fusione (per creare e riconciliare i frammenti rispettivamente). Inoltre per preservare la consistenza bisogna garantire condizioni di consistenza sugli interi oggetti forniti dalle applicazioni. Queste condizioni includono operazioni concesse, vincoli dei loro valori di ingresso e condizioni dello stato degli oggetti.

Reporting non propone nuovi modi per garantire la consistenza, ma sottotransazioni possono essere legate a transazioni compensative, eccetto per le non compensabili, per mantenere la consistenza semantica in caso di abort.

4.2.2 Sommario

La tabella seguente sintetizza i principali concetti usati per preservare la consistenza. L'informazione semantica degli oggetti è essenziale per garantire la consistenza nelle applicazioni mobili.

Tutti i lavori analizzati utilizzano la semantica degli oggetti in modi differenti.

Clustering definisce gradi di inconsistenza basati sulle applicazioni semantiche.

Two-tier replication introduce un criterio di accettazione per transazioni di tentativo e base.

HiCoMo genera transazioni base da transazioni HiCoMo commutative.

IOT usa applicazioni specifiche risolutrici per riconciliare la seconda classe di transazioni.

Pro-motion utilizza informazioni semantiche per costruire compact mentre *Semantic-based* le usa per dividere e unire oggetti.

Reporting basa la delega su requisiti semantici, e *Prewrite* definisce varianti dei dati (oggetti prewrite/write) “semanticamente identiche”.

	Consistency and semantic information
Clustering	Definition of the function *h* and degrees of inconsistency
Two-tier replication	To define commutative tentative transactions and acceptance criteria
HiCoMo	To define: aggregate tables, commutative HiCoMo transactions, and error margins. To execute the transaction transformation function
IOT	ASR resolution conflicts
Pro-motion	Compacts construction (type specific methods, consistency rules and obligations) and contingency procedures
Reporting	Delegation, compensating transactions
Semantics-based	Objects fragmentation (consistency conditions and split/merge operations)
Prewrite	Definition of data variants (prewrite/write)

Figura 14 – Sintesi sui vari aspetti della consistenza

4.3 Isolation

In questa sezione si discutono tre problematiche concernenti l'isolamento:

1) il grado di visibilità per le transazioni che hanno eseguito il commit localmente;

2) scelta dei protocolli di controllo della concorrenza;

3) consistenza mutua per i dati replicati (*one-copy serializability*).

4.3.1 Aspetti di visibilità

Per quanto concerne la visibilità dei risultati delle transazioni intermedie, *Clustering, Two-tier replication, HiCoMo, IOT, Pro-motion e Semantics-based* danno visibilità dei risultati localmente committati alle transazioni locali sullo stesso MH. Prewrite rende pubblici i risultati dei commit locali quando il commit locale è riportato su BS.

In *Reporting* la visibilità è permessa a transazioni atomiche, *reporting* e *co-transaction* ma non a transazioni non compensabili.

Una transazione atomica può fare commit persino prima del commit della sua transazione genitrice, e le sue modifiche sul database divengono visibili alle altre transazioni. L'obiettivo di *reporting* e *co-transaction* è precisamente di permettere la visibilità dei risultati parziali mentre sono in esecuzione.
Prendendo *Pro-motion* e *Reporting* come transazioni *open-nested*, l'isolamento globale non è garantito visto che le sottotransazioni non sono eseguite isolatamente. Dopo il processo di sincronizzazione, Pro-motion divide la sua *long-lived transaction*.Tutte le operazioni sincronizzate con successo costituiscono una transazione separata che esegue il commit sul database server. I risultati di queste *split transaction* (che hanno eseguito il commit) sono visibili al completo ambiente del database.

4.3.2 Schemi per il controllo della concorrenza

Per manipolare esecuzioni concorrenti ,*Clustering* e *Prewrite* utilizzano protocolli orientati al 2PL e propongono una nuova tabella dei conflitti.

- *Clustering* utilizza il 2PL stretto e propone quattro tipi di lock che corrispondono a operazioni *weak* e *strict* (WR,WW,SR,SW). Quattro tabelle dei conflitti sono proposte per la compatibilità dei lock La funzione di proiezione h utilizza le tabelle di conflitto per riportare le operazioni *strict* sulle versioni *weak* dei dati in dipendenza dei requisiti di consistenza delle applicazioni.

  Per esempio la consistenza richiede di rendere una scrittura *strict* su un oggetto una scrittura *strict* su tutte le sue copie. Di conseguenza un SW blocco è incompatibile con ogni altro blocco. Transazioni *weak* rilasciano i loro lock al commit locale e le transazioni *strict* al commit. Se le transazioni *strict* e *weak* sono eseguite in maniera concorrente su un *cluster*, uno schedule corretto assicura che un'operazione di lettura *weak* legga i dati modificati dalla precedente operazione di scrittura (*weak* o *strict*) e un'operazione di lettura *strict* legga i dati modificati dall'ultima scrittura *strict*.

- *Prewrite* usa un protocollo di 2PL e la tabella delle operazioni in conflitto include le *preread* e le *prewrite* (PR,PW,R,W).
  I blocchi relativi a *prewrite* e *preread* riguardano la versione dei dati *prewrite*.
  I blocchi relativi a letture e scritture riguardano la versione dei dati *write*.
  Tutti i blocchi sono gestiti dalla BS. Per rendere le *prewrite* permanenti i *prewrite locks* devono essere convertiti in *write lock*; in questo modo il gestore dei dati può scrivere e fare il commit delle transazioni mobili. I *preread lock* sono rilasciati al momento del commit locale mentre i blocchi relativi a *prewrite/write/read* sono effettuati al momento del commit.
  Usando oggetti semplici (senza le due varianti) le *prewrite* sono identiche alle *write* e l'algoritmo comporta l'utilizzo di un 2PL rilassato. *Prewrite* assicura che l'algoritmo di elaborazione va avanti senza blocchi basati su un protocollo che produce solo schedule serializzabili. La serializzabilità è basata sul commit locale permesso alle transazioni mobili.

- In *HiCoMo*, dato che le transazioni HiCoMo sono commutative, la loro esecuzione può avvenire senza restrizioni forti sull'ordine. Dato che le transazioni base non sono commutative, perché possono eseguire operazioni di divisione o moltiplicazione, l'ordine tra esse e le transazioni HiCoMo è importante.
  E' utilizzato un controllo della concorrenza ottimistico basato sui *timestamp* [22] per evidenziare i conflitti.
- Analogamente,in *Two-tier replication* se le *tentative transaction* sono commutative non c'è la necessità di un meccanismo di controllo della concorrenza locale. Tuttavia per l'esecuzione delle *base transaction* è utilizzato un meccanismo di locking.
- In *IOT*, il controllo della concorrenza è fatto a due livelli. Tra client, il controllo della concorrenza globale è fatto utilizzando un controllo della concorrenza ottimistico (OCC) [23]. All'interno di un client, il controllo della concorrenza locale è garantito con un 2PL stretto insieme ad un periodico controllo dei deadlock. La serializzabilità è garantita localmente.

– Poichè in *Pro-motion* il progettista del compact può determinare criteri di correttezza e metodi per il controllo della concorrenza per compact, propone di usare una scala a dieci livelli. I livelli sono caratterizzati dal fatto di basarsi sui gradi di isolamento definiti nell'ANSI SQL standard [24]. Il livello 9 rappresenta un'esecuzione seriale delle transazioni e il livello 8 un'esecuzione serializzabile. Ogni livello successivo rappresenta un grado più debole di isolamento. Al livello 0 non c'è alcuna garanzia di isolamento. Poiché l'uso arbitrario dei livelli può portare ad inconsistenze, *Pro-motion* propone semplici regole:

  1) Le transazioni impongono un livello minimo per le operazioni di lettura e scrittura.
  2) Ogni operazione è associata ad un livello.
  3) Nessuno dei livelli delle operazioni di scrittura è più basso rispetto al livello di scrittura della transazione.
  4) Nessuno dei livelli delle operazioni di lettura è più basso rispetto al livello di lettura della transazione.
  5) Il più basso livello di ogni operazione di lettura è superiore o uguale al più alto livello richiesto da ogni operazione di scrittura.

– In *Semantic-based*, per assicurare la serializzabilità, le transazioni locali hanno accesso ai frammenti memorizzati utilizzando i protocolli per il controllo della concorrenza convenzionali (esempio 2PL).

4.3.3 Problemi di replicazione

Problemi relativi alla replicazione sono legati alla gestione delle transazioni mobili.
Clustering e *Two-tier replication* mantengono due versioni dei dati. Entrambe le versioni sono localizzate sul MH, una di esse (*weak/tentative*) è utilizzata per supportare l'evoluzione dei dati in modalità disconnessa. L'altra versione (*strict/master*) deve essere sempre consistente. La consistenza nella versione *strict/master* è preservata usando il metodo di *one-copy serializability*. Per

provvedere alla coerenza *Clustering* utilizza un *quorum consensus* e *Two-Tier Replication* utilizza un protocollo di replicazione *lazy-master*.In quest'ultimo caso le versioni *tentative* dei dati sono scartate a riconnessione avvenuta perché non sono aggiornate rispetto alle versioni *master*.

HiCoMo, Pro-motion e Prewrite considerano un differente approccio.

Costruiscono un tipo particolare di dato (dalle sorgenti memorizzate sui FH) che sarà memorizzato sui MH e può essere considerato una specie di replica.

In *HiCoMo*, tabelle aggregate sono generate da tabelle base. Il criterio di correttezza è detto "convergence" dove le tabelle base eventualmente riflettono gli aggiornamenti fatti nelle tabelle aggregate.

L'approccio è simile in *Prewrite*, dove la "variante prewrite " è una variante ridotta del "valore write".

In *Pro-motion*, diversamente da *HiCoMo* e *Prewrite*, i *compact* contengono non solo dati specifici ma anche speciali informazioni per utilizzarli.

La flessibilità offerta dai *compact* permette a *Pro-motion* di supportare alcuni schemi dinamici di replicazione con una varietà di vincoli di consistenza e criteri di correttezza.

In *IOT*,è utilizzata una variante di *Read-One,Write-All* (ROWA) è usato per mantenere la consistenza in un ambiente totalmente connesso. Con ROWA la prima classe di transazioni sono serializzabili con tutte le transazioni che hanno eseguito il commit.

In modalità disconnessa, si assume un'evoluzione ottimistica sui MH.

IOT definisce dei criteri di correttezza detti *global certifiability*. Essi richiedono che una transazione pendente sia globalmente serializzabile con e dopo tutte le transazioni che precedentemente hanno eseguito il commit. La certificazione globale è forzata con una riesecuzione semantica della seconda classe di transazioni pendenti. Questo è il criterio di consistenza di default.

4.3.4 Altri approcci per il controllo della concorrenza

In questo paragrafo vengono presentati approcci alternativi per il controllo della concorrenza ed in particolare ottimistici, ovvero i blocchi in lettura sono garantiti

su richiesta e quelli in scrittura sono differiti fino al momento del commit. La risorsa può assumere valori differenti su diversi host mobili.

4.3.4.1 O2PL-MT (O2PL per Mobile Transaction)

Il 2PL non è opportuno per l'esecuzione delle transazioni distribuite che coinvolgono i MH. Questo perchè non è noto il tempo di blocco delle risorse per via dell'impredicibilità delle disconnessioni. Sono state proposte delle varianti in cui uno schema di controllo della concorrenza integra approcci pessimistici ed ottimistici. Viene associato un timeout al blocco dei dati. Questo è il tempo stimato entro il quale la transazione richiede il commit. Se la richiesta non avviene all'interno di quel periodo (a causa di disconnessioni) allora la politica pessimistica muta in ottimistica. Alla riconnessione le transazioni che ottimisticamente avevano fatto il commit sono rieseguite.
O2PL-MT (O2PL per Mobile Transactions) estende l'algoritmo ottimistico del 2PL (O2PL) agli ambienti mobili. In questo algoritmo blocchi in lettura sono garantiti su richiesta e quelli in scrittura sono differiti fino al momento del commit. Lavorando in un contesto replicato, l'algoritmo O2PL-MT riduce il numero di messaggi da inviare quando si rilasciano i blocchi in lettura. O2PL-MT permette di realizzare un blocco in lettura, per una risorsa qualsiasi, su ciascuna sua copia per sito. Ciò senza curarsi del fatto che la risorsa su un sito è in uno stato differente rispetto alla copia sul sito dove è impostato il blocco.

4.3.4.2 ASGT

L'alto tasso di blocco e l' alta probabilità di abort, per via dell'instabilità della larghezza di banda e della mobilità degli utenti, portano ad un nuovo metodo quale *ASGT* (Active Serialization Graph Technique) [34] che determina un tasso di blocco molto basso e può abortire le transazioni che si prevede essere coinvolte in scheduling non serializzabili. In *ASGT*, variante di *SGT*, l'operazione di lettura non ostruirà il funzionamento di scrittura, in modo che la concorrenza può essere notevolmente migliorata. E' possibile effettuare l'abort delle transazioni coinvolte in uno scheduling non seriablizzabile per ridurre la coda di conflitto e per ridurre il costo del rollback. Grazie alla storia dello scheduler mantenuta a tempo di esecuzione metodi come *MDWL* (Modified Waiting-Depth Limited)

relativi alla politica di abort possono essere integrati con elasticità in *ASGT* per migliorare le prestazioni del sistema e per fare diminuire il costo di programmazione per le varie applicazioni. Il costo di programmazione di *ASGT*, derivato da *SGT*, è molto più basso di quello di *SGT* stesso.

I risultati sperimentali e l'analisi teorica hanno dimostrato che *ASGT* è apparentemente superiore ad un 2PL migliorato (I2PL) nell'ambiente mobile. *ASGT* permette l'esecuzione delle transazioni forti e la serializzabilità dello scheduling.

SGT è un metodo di controllo di concorrenza che può mantenere l' ordine esplicito di conflitto a tempo di esecuzione. In *SGT*, i conflitti fra le transazioni sono mantenuti in un grafo, denominato *StoredSG* (grafo delle serializzazioni), SSG in breve. Differentemente da uno SG che contiene soltanto le transazioni attive, uno SSG include sia tutte le transazioni attive che alcune transazioni che devono eseguire il commit.

Questo serve perché in SSG l'operazione di lettura non blocca l'operazione di scrittura per cui tenere traccia delle risorse bloccate da transazioni che hanno eseguito il commit è importante perché transazioni attive possono aver letto dati da quelle transazioni che hanno eseguito il commit solo successivamente. Per cui si possono scoprire le letture inconsistenti e quindi si individuano i conflitti mancanti.

SGT può trarre vantaggio notevole dal basso tasso di blocco perché l'operazione di lettura non ostruirà mai l'operazione di scrittura. Nello scheduling, tuttavia, occorre maggiore memoria per memorizzare più informazioni (infatti effettuo più operazioni e quindi memorizzo più dati).

Una riflessione profonda indica chiaramente che l'unico scopo di mantenere delle transazioni che hanno eseguito il commit è impedire allo scheduler la produzione di uno schedule non serializzabile a causa dei conflitti mancanti. Grazie al grafo di serializzabilità esplicito mantenuto a tempo di esecuzione, si è sviluppato un nuovo schema che può controllare le transazioni che sono in conflitto e predire se formano o meno un ciclo.

4.3.4.3 DC/POS-PAI-2PL (Divergence control/Prudent order sharing – avoiding priority inversion -2PL)

Nei sistemi di database in tempo reale, l' obiettivo del controllo della concorrenza potrebbe richiedere:

- un sistema di salvataggio delle risorse;
- il supporto delle disconnessioni;
- la garanzia della consistenza della base dati;
- la riduzione del numero di transazioni abortite;
- l'esclusione dell'inversione della priorità.

A tale proposito vanno considerati:

- il modello delle transazioni in tempo reale innestate;
- un modello di controllo dei blocchi basato sulla condivisione prudente e ordinata;
- un protocollo di Check-Out/Check-IN che supporta le disconnessioni.

DC/POS-PAI-2PL è una strategia di lock a due fasi che integra questi metodi per transazioni mobili in tempo reale. Analizzando le prestazioni di *DC/POS-PAI-2PL* è mostrato come il modello suggerito possa migliorare le prestazioni del sistema rispetto al 2PL standard.

Si tratta di un protocollo di concorrenza basato sui blocchi, legati a questo modello di transazioni innestate.

Nelle strategie di controllo della concorrenza basate sui blocchi delle risorse i conflitti sono risolti attraverso l'arresto di una delle transazioni che accede a risorse già detenute da un'altra transazione . I blocchi potrebbero crescere insieme al ritardo di esecuzione delle transazioni in tempo reale su rete wireless.

Quindi si sviluppa un nuovo modello che può ridurre i blocchi ed assicurare che più MRTT possano essere ultimate all'interno delle deadline.

In generale nei modelli di lock esclusivo (X) le altre transazioni non possono accedere all'oggetto fin a che il blocco non è rilasciato. Apparentemente questo modello di blocco non è sufficiente alle MRTT perchè molte transazioni mobili in tempo reale sono delle *query transaction*.

Per transazioni in tempo reale è auspicabile avere in tempo dati parziali piuttosto che informazioni completamente corrette ma non in tempo.

Quindi il modello base di blocco si estende come segue:

1) Le transazioni mobili in tempo reale sono divise in QT (*query transaction*) e UT (*update transaction*). Le prime bloccano i dati con *query lock* (solo lettura) e le UT bloccano i dati con blocchi in lettura (potrebbe essere necessaria una modifica in seguito) o blocchi in scrittura in accordo ai tipi di operazioni.

2) Le relazioni tra QT e UT includono compatibilità (Y) , incompatibilità (N) e compatibilità limitata (LY).

LY potrebbe essere:

- LY-1 indica che le QT sono abilitate a leggere dati sporchi modificati da UT;
- LY-2 indica che le UT sono abilitate a modificare i dati letti dalle QT.

3) Le relazioni tra UT includono compatibilità (Y) , incompatibilità (N) e compatibilità ordinata (dette OY).

La compatibilità ordinata si ha quando le transazioni con bassa priorità sono bloccate da transazioni con alta priorità. In più per permettere gli abort a cascata non è permesso ad un blocco R (di read) di essere ordinatamente compatibile con uno W (di write). Il modello di lock esteso descritto sopra è detto modello di blocco per il controllo della divergenza basato su una condivisione prudente e ordinata. La matrice dei blocchi compatibili è mostrata in figura. Il grado di concorrenza nel modello è più alto che nei modelli base di blocco mentre il criterio di serializzabilità stretta è stato rilassato attraverso la compatibilità limitata e ordinata. Perciò il modello è appropriato per i sistemi di database in tempo reale.

T_r \ T_h	Q	R	W
Q	Y	Y	LY-1
R	Y	Y	N
W	LY-2	N/OY	N/OY

Figura 15 – Matrice di compatibilità del modello di controllo della divergenza con condivisione prudente e ordinata

Per assicurare la consistenza del database, le MRTT dovrebbero obbedire ai controlli di divergenza ed alle regole di condivisione ordinata. La serializzabilità

dei conflitti tra QT e UT può essere rilassata per il controllo della divergenza. La chiave del controllo della divergenza è l'abilitazione di regole di inconsistenza basata sulle soglie:

- di accumulazione dell'import (**ImAC**) dell'inconsistenza;
- di accumulazione dell'export (**ExAC**) dell'inconsistenza.

Abbiamo le seguenti regole di controllo della divergenza:

- Ogni volta che una QT richiede un *Q lock* con la relazione LY-1, sia il QT ImAC che l' ExAC in conflitto dell'UT crescono di un'unità e quindi si verifica se esse hanno ecceduto le loro soglie. Se no la richiesta è abilitata altrimenti respinta.

- Ogni volta che una UT richiede un blocco in scrittura con la relazione LY-2, tutte le ImAC delle QT in conflitto aggiungono uno e le ExAC delle UT crescono del numero totale delle QT in conflitto con essa, e quindi si verifica se esse hanno ecceduto le soglie. Se no la richiesta è abilitata altrimenti respinta.

Le regole di rilascio dei blocchi sono particolarizzate per le transazioni LY ed OY e viene definita una relazione di blocco ordinato OS tra transazioni.
Se si scrive OS(Ti, Tj , x), quindi Tj non può realizzare nessun blocco prima che Ti abbia rilasciato tutti i blocchi. Si dice che Tj attende ordinatamente Ti e quindi Tj può eseguire il commit dopo che Ti finisce.
Quindi una transazione in tempo reale pronta per eseguire il commit, se soddisfa le regole citate, può eseguirlo; altrimenti se arrivano le loro deadline il sistema esegue l'abort di tutte le transazioni precedenti che hanno una relazione di blocco ordinato OS con essa e lascia che esegua il commit. Se essa non incontra la regola di rilascio ordinato dei blocchi allora attende che arrivi la sua deadline.
La strategia del 2PL è una condizione sufficiente per assicurare la serializzabilità delle transazioni correnti, ed è diventata un protocollo standard basato sui blocchi. Quindi i protocolli mostrati sopra potrebbero anche conformarsi con esso, vale a

dire, tutte le operazioni di blocco di ogni transazione devono essere precedute da tutte le operazioni di unlock.
Inoltre in tale approccio si danno delle regole per risolvere i conflitti che prendono in considerazione la struttura innestata delle relazioni tra MRTT. Queste si basano sul concetto di famiglia di transazioni e le regole si particolarizzano in base alle relazioni tra transazioni e al fatto che le transazioni possono o meno appartenere alla stessa famiglia.

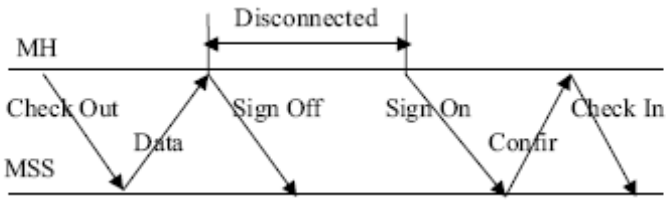


Figura 16 – Protocollo di check Out/check In

4.3.4.4 DHP 2PL (distribuited high priority 2PL) basto sulla similarità

La similarità è vicina relativamente all'importante idea di calcolo impreciso per i sistemi in tempo reale e all'idea di calcolo parziale per i database. E' stato mostrato essere molto valida nel migliorare le prestazioni di un RTDBS (REAL TIME DATABASE SYSTEM) [35,36]. Per molte applicazioni in tempo reale, il valore dell'oggetto che ne modella uno in ambiente esterno, in generale non può essere aggiornato continuamente per tenere traccia perfettamente delle dinamiche degli oggetti esterni. Allo stesso tempo è anche non necessario per il valore dei dati essere perfettamente aggiornati o precisi per essere utili.
In particolare il valore dei dati che sono lievemente differenti sono spesso intercambiabili come dati letti per le transazioni. Questa osservazione è alla base del concetto di similarità tra dati.
Questo concetto di similarità può essere usato per estendere i convenzionali criteri di correttezza per il controllo della concorrenza in RTDBS, e per bilanciare la precisione dei dati (basandosi sulla similarità) e il carico di lavoro. Le nozioni di similarità e di similarità forte all'origine introdotte in letteratura hanno la

proprietà di scambiare operazioni simili in conflitto nello schedule per preservare sempre la similarità in uscita. Se due operazioni nello schedule sono fortemente simili (sono sia letture che scritture, e il valore dei dati coinvolti sono fortemente simili) possono essere sempre utilizzate in modo intercambiabile nello schedule senza violare l'integrità e la consistenza della base dati. Tutte le relazioni di similitudine considerate sono relazioni di similarità forte, e tutti i dati simili possono essere usati in modo intercambiabile nello schedule senza per questo giungere a risultati scorretti.

Il metodo *DHP 2PL* propone di adottare un approccio basato sui lock in cui è usato sia il riavvio delle transazioni che l'eredità delle priorità per risolvere il problema dell'inversione delle priorità.

Estendendo il ben noto *HP-2PL* ,una estensione distribuita di *HP-2PL*, si ottiene *DHP-2PL* che è proposta per MDRTDBS.

Nel protocollo particolare attenzione è stata prestata alle caratteristiche dell'ambiente mobile come la ristrettezza di banda e le frequenti disconnessioni.

Ad esempio è stata posta particolare attenzione su come ridurre il numero di transazioni riavviate che possono essere molto costose a causa della limitatezza di banda.

Se una transazione sta eseguendo il commit non potrà essere riavviata anche se ha un lock conflict con una transazione a più alta priorità.

L'eredita della priorità è usata per ridurre il tempo di blocco delle transazioni ad alta priorità.

DHP-2PL è un protocollo di blocco distribuito. Il sistema del database locale su ogni base station ha uno scheduler dei lock, che manipola le richieste di lock per gli oggetti che si trovano sulla base station. Segue la definizione del *DHP-2PL* , dove Tr e Th sono le transazioni richiedente il lock e detenente il lock rispettivamente:

```
Lock Conflcit (Tr, Th)
Begin
        If      Priority(Tr) > Priority(Th)
                If      Th is not committing
                        If      Th is a local transaction
                                Restart Th locally
                        Else
                                Restart Th globally
                        Endif
                Else
                        Block Tr until Th releases the lock
                        Priority(Th):= Priority( Tr) + fixed priority level
                Endif
        Else
                Block Tr until Th releases the lock
        Endif
End
```

Figura 17 – Algoritmo atto a risolvere l'inversione della priorità

Il seguente algoritmo sintetizza come lo schema di attesa cautelativa è incorporato in DHP-2PL:

```
If      (the priority of the lock-requester > the priority of the lock-holder )  and
        (the lock-holder is not committing)
        Restart the lock-holder (globally or locally, depending on the type of the transaction)
Else
        If      location indicator of the lock-holder is "mobile client"
                If      the time already spent at the client side > threshold
                        Ping the mobile client where the lock-holder is residing
                        //* the base station sends a message to the mobile client
                        to test whether the mobile client is disconnected or not *//
                        If      no response from the mobile client
                                Restart lock-holder
                        Else
                                Block the lock-requester
                        //* repeat the checking after another threshold *//
                        Endif
                Else
                        Block the lock-requester
                        //* the checking will be performed again when the time already spent at the
                        client side is greater than the threshold value *//
                Endif
        Else
                Block the lock-requester.
        Endif
Endif
```

Figura 18 – Algoritmo dello schema di attesa cautelativa

La soglia è un parametro accordabile. E' funzione della prestazione media della rete mobile nelle situazioni di utilizzo normale. Se una transazione è stata su un

cliente mobile per lungo tempo, per esempio più grande rispetto al valore della soglia, e la sua base station non può comunicare con l'utente correntemente, si assume la disconnessione. La transazione che detiene il blocco dovrebbe essere riavviata anche se la sua priorità è più alta rispetto alle transazioni che richiedono il blocco. Sebbene riavviare le transazioni proprietarie del blocco potrebbe portare ad un'elevata probabilità di perdere la deadline, il riavvio non influenza le prestazioni del sistema significativamente visto che la transazione proprietaria del blocco è come se perdesse la deadline per via della disconnessione. Riavviare le transazioni proprietarie dei blocchi fa crescere la possibilità di incontrare le deadline delle transazioni che richiedono il blocco. Altrimenti c'è la possibilità che alcune di esse perdano la deadline. Notare che l'assunzione basata sulla disconnessione e l'interrogazione del cliente mobile potrebbe non essere sufficienti visto che le reti mobili sono soggette a differenti fallimenti transitori delle comunicazioni, i quali sono molto meno dannosi rispetto alle disconnessioni. Un fallimento transitorio della comunicazione dura per un tempo brevissimo e può essere risolto di solito con la ritrasmissione dei dati.

Le transazioni riavviate in MDRTDBS (Mobile distributed real-time database system) possono essere molto costose e potrebbero avere un'alta probabilità di perdere le loro deadlines. Per ridurre la probabilità dei conflitti dei dati e il riavvio delle transazioni, una nozione meno restrittiva dei criteri di correttezza potrebbe essere esplorata per il controllo della concorrenza. In particolare il concetto di similitudine, che si basa sul tempo di validità dei dati, è mostrato effettivamente per RTDBS. Si incorpora il concetto di similarità in *DHP-2PL* sia per far crescere la concorrenza del sistema che per ridurre il numero di transazioni riavviate e bloccate per via dei conflitti sui dati. Il nuovo protocollo è detto *SDHP 2PL* (Similarity based DHP 2PL).

Sebbene il concetto di similitudine per il controllo della concorrenza in RTDBS non è nuovo, molti problemi tecnici nel progetto di un protocollo *SDHP 2PL* sono ancora non chiari per esempio come risolvere i conflitti sui blocchi tra le operazioni di lettura e di scrittura quando alcuni conflitti sono tra transazioni che lavorano su dati simili e altri no.

La strategia base per la soluzione dei conflitti nel protocollo *SDHP-2PL* può essere sintetizzata come segue:

Si supponga che la transazione Ti emetta una richiesta di blocco in lettura sul dato Dk, e Dk sia già bloccata in scrittura da un insieme di transazioni TH. Sia Tj la più recente delle transazioni che ha eseguito il commit e che abbia aggiornato Dk. Se l'operazione di scrittura su Dk di Tj e le operazioni di scrittura di tutte le altre transazioni di TH sono simili, allora la richiesta di blocco in lettura sull'oggetto Dk potrebbe essere garantito. Questo perché Ti potrebbe sempre leggere da valori dei dati simili, non preoccupandosi di quali transazioni in TH eseguono il commit (o aggiornamento di Dk) per prime.
Si supponga che Ti richieda un blocco in scrittura su Dk, e Dk sia già bloccato in scrittura da un insieme di transazioni TW e bloccate in lettura da un insieme di transazioni TR. Sia Tj la transazione che più recentemente abbia eseguito il commit aggiornando Dk. Ci sono due casi da considerare:

- Se TR è vuoto, e le operazioni di scrittura in conflitto di Ti e le operazioni di scrittura in conflitto di tutte le transazioni in TW sono simili, allora il blocco in scrittura di Ti potrebbe essere concesso. Ciò perché lo stato finale del database potrebbe essere simile disinteressandosi di quali operazioni di scrittura si elaborano prima.
- Se TR è non vuoto, e le operazioni di scrittura in conflitto di Ti, le operazioni di scrittura in conflitto di Tj, e le operazioni di scrittura in conflitto di tutte le transazioni in TW sono simili, allora il blocco in scrittura di Ti potrebbe essere concesso. Ciò perché lo stato finale del database potrebbe essere simile e le operazioni di lettura potrebbero leggere da valori simili dei dati senza preoccuparsi di quali operazioni elaboro prima.

Ovviamente se il test di similarità di sopra alla richiesta di lock fallisce, allora due alternative vanno considerate:

- la richiesta di blocco della transazione potrebbe essere bloccata;
- le transazioni che causano il blocco dovrebbero essere riavviate.

Quali alternative potrebbero essere usate dipende dalle proprietà delle transazioni in conflitto. In SDHP-2PL adottiamo entrambe le politiche come segue:

Sia CT un sottoinsieme di TH che consiste di transazioni che bloccano Dk in modo conflittuale e non simile alle richieste di blocco di Ti, e SCT sia un sottoinsieme di CT che consiste in transazioni con priorità bassa rispetto a Ti. Se il riavvio di tutte le transazioni in CT potrebbe implicare il passaggio del test di similitudine da parte di Ti e le priorità di tutte le transazioni in CT sono basse rispetto a Ti, allora le transazioni in CT vengono riavviate, e la richiesta di blocco di Ti è concessa.

Se alcune transazioni in CT hanno priorità più alta rispetto a Ti, per esempio SCT non è uguale a CT, semplicemente riavviando tutte le transazioni in SCT potrebbe non essere risolto il problema del conflitto sul blocco. Riavviare tutte le transazioni in CT includendo le transazioni a più alta priorità potrebbe portare a un problema di cicli nei riavvii quindi il riavvio delle transazioni ad elevata priorità potrebbe far riavviare Ti in seguito.

Ci sono due possibili soluzioni per risolvere il problema di sopra. Per primo si potrebbe utilizzare un approccio aggressivo nel quale Ti viene bloccata quando CT non è uguale a SCT e tutte le transazioni in SCT riavviate.

La ragione del riavvio delle transazioni in SCT è di minimizzare il tempo di blocco di Ti poiché il tempo richiesto per completare le transazioni a più bassa priorità potrebbe essere molto lungo specialmente in un'inaffidabile rete mobile.

Ogni volta che una transazione in (CT-SCT) realizza il blocco su Dk, un test di similarità potrebbe essere eseguito per Ti. In questo modo, presto o tardi, Ti setta un blocco su Dk quando tutte le transazioni a più alta priorità in CT hanno rilasciato il blocco su Dk.

Per prevenire ripetuti riavvii di transazioni a bassa priorità, a una transazione non è permesso di ottenere il blocco se una transazione a più alta priorità, che non è simile ad essa, sta attendendo per il blocco.

Nell'approccio aggressivo, riavviando le transazioni a più bassa priorità, il tempo di blocco delle transazioni ad elevata priorità può essere minimizzato. Comunque il costo è il riavvio di molte transazioni e ciò non è desiderabile in ambiente mobile dato che una transazione riavviata ha un'elevata probabilità di perdita di deadline. Il secondo metodo è l'approccio conservativo, nel quale le transazioni che richiedono i blocchi sono bloccate se CT-SCT è non vuoto, per esempio esiste una transazione a più alta priorità in CT. Ogni volta che una transazione rilascia il blocco su Dk, un test di similarità è eseguito per Ti. Ti è abilitata a settare un

blocco su Dk quando tutte le transazioni a più alta priorità proprietarie di blocchi sono simili ad essa. Il numero di transazioni riavviate sotto l'approccio conservativo potrebbe essere più piccolo rispetto a quelle dell'approccio aggressivo. Comunque il tempo di blocco delle transazioni a più alta priorità potrebbe essere lungo e si pone un problema di rinvio indefinito.

4.3.4.5 Concurrency Control Mechanism (CCM) per Mobile Database Systems (MDS)

Una forma più debole di consistenza è desiderabile e si presenta per questo un *CCM* che usa le proprietà semantiche per la comunicazione dei dati.

Nell'elaborazione dei dati un criterio di correttezza più debole è accettabile in un certo numero di situazioni ed è usata la *ε – serializability* [33], che tollera una quantità limitata di inconsistenze specificate da ε per sviluppare il modello CCM. È basata su uno schema a due stadi di replicazione che producono uno schedule ε serializzabile .

L'idea di base della replica a due stadi è prima quella di permettere che gli utenti facciano funzionare le transazioni sulla MU. Gli aggiornamenti dei dati localmente si hanno quando la MU si collega ai server, allora queste transazioni sono eseguite ancora dai server come *transazioni base (BT)*. *Le BT* sono serializzate sulla copia principale dei dati e le MU sono informate circa il fallimento delle BT. Ma il problema con questo metodo è che la MU esegue la transazione senza la conoscenza di che cosa altre transazioni stanno facendo, il che può condurre a tantissime transazioni abortite. Un altro problema è che il periodo di commit di una transazione su MU è grande perché conosce il relativo risultato solo dopo che la BT è stata eseguita. In questo approccio si modifica lo schema a due stadi della replica per ridurre il numero di abort e il tempo di commit.

Supponiamo che c'è un server centrale che tiene e controlla la base di dati. Sia D un insieme di elementi dei dati tali che D = $\{D_i\}$, dove i appartiene ad N , insieme dei numeri naturali e D_i appartiene ad S dove S è uno spazio metrico. Sia d_i il valore corrente di D_i. Gli elementi dei dati sono replicati sulle MU e sia n_i il numero di repliche di Di in MDS. Manteniamo un limite per la quantità di

cambiamenti che può essere fatta sulla replica di ogni MU, così Λ_i denota i massimi cambiamenti totali permessi su una replica di D_i su una MU. Se la transazione cambia il valore di dati non più di Λ_i in un MU, esegue il commit senza problemi; non deve attendere i risultati della BT del DBS. Ciò riduce il tempo di commit per le transazioni ed inoltre contribuisce a ridurre il numero di abort, dato che la BT non può eseguire il commit. Per controllare la validità del Λ_i, definiamo un parametro di timeout il cui il valore indica una durata in cui il valore di Λ_i è valido.

I valori di timeout dell'elemento dei dati dovrebbero essere un certo (I) multiplo di tempo di ciclo di broadcast (T). *I* dipende dalla frequenza degli aggiornamenti ricevuti per l'elemento dei dati e anche dovrebbe essere sufficientemente grande in modo che MU trasmetta i suoi aggiornamenti nel tempo $I \times T$. Il server non aggiornerà il valore dei dati fino a che $I \times T$ non trascorra. Supponiamo che le MU considerino il tempo di *uplink* e trasmettano i loro aggiornamenti al server prima del time-out. Il client si può staccare dal MDS durante il periodo di prespegnimento e può effettuare gli aggiornamenti. Se il cliente sta disconnesso per un periodo più lungo del time-out, allora quando si ricollega dovrebbe leggere i nuovi valori di Λ. Se gli aggiornamenti sono all'interno dei nuovi limiti settati da Λ allora la MU può trasmettere gli aggiornamenti al server altrimenti la MU dovrà ostruire alcune transazioni in modo che gli aggiornamenti totali siano all'interno di Λ. Le transazioni ostruite dovranno attendere fino a che i nuovi valori di Λ non arrivino al MU.

Il meccanismo per mantenere l'ESR consiste nel *controllo di divergenza* (CC) e nel ripristino della *consistenza*. Una transazione importa l'inconsistenza leggendo dati di altre transazioni che non hanno ancora eseguito il commit. Una transazione esporta l' inconsistenza permettendo che l'altra transazione legga i relativi dati che non sono ancora "committati" . Le transazioni hanno un *import counter* ed un *export counter* dell'inconsistenza in analogia al metodo *DC/POS-PAI-2PL*.

4.3.5 Discussioni

La computazione in ambiente mobile di solito coinvolge alcuni tipi di replicazioni visto che i dati memorizzati sui MH sono estratti da database localizzati su rete

fissa. Riguardo alla replicazione si considerano due livelli di correttezza: locale (su ogni host) e globale (su tutti gli host). In un ambiente completamente connesso è possibile garantire la correttezza globale (o consistenza mutua) del sistema mobile. Per esempio, vedi Clustering, Two-tier replication e IOT.

In modalità disconnessa o debolmente connessa i criteri di correttezza globale devono essere rilassati per evitare il blocco dei MH così come le esecuzioni sul FH. Così l'evoluzione locale dei dati potrebbe continuare dando ai MH autonomia. Eventuali correttezze globali [25,26] sembrano adeguate agli ambienti mobili perchè la consistenza è ottenuta o richiesta "in uno specifico real-time","in un intervallo di tempo" oppure "dopo che si è raggiunta una certa soglia per un dato valore". Tra i modelli analizzati notare che in *Clustering*, una eventuale consistenza è proposta per definire gradi di inconsistenza (4.2.1). Ciò permette "un massimo numero di transazioni che possono operare in modalità disconnessa", "un intervallo di valori accettabili che i dati possono assumere","un massimo numero di copie divergenti per dato" e "un massimo numero di aggiornamenti per dati non riportati su tutte le copie". Come possiamo notare nella tabella seguente, sebbene i risultati locali sono di tentativo (alla riconnessione un commit globale va fatto) la maggior parte dei modelli analizzati li rende localmente visibili. E' anche interessante notare che in generale la correttezza locale è assicurata usando approcci tradizionali di blocco (esempio 2PL). Tuttavia il protocollo di 2PL non è adeguato per l'esecuzione delle transazioni mobili distribuite e si stanno sviluppando protocolli ottimistici. (vedi sez. 4.3.4).

	Visibility	Concurrency control	Replication issues
Clustering	*local committed* transaction results are visible to local weak transactions on the same MH	2PL, 4 conflict tables and new lock types are proposed	2 versions of data: strict (one-copy serializability using quorum consensus protocol), weak (degrees of inconsistency, data evolution in disconnected mode)
Two-tier replication	*local committed* transaction results are visible to local tentative transactions on the same MH	Locking mechanisms for base transactions	2 versions of data: master (one-copy serializability using a lazy master replication protocol), tentative (local data evolution in disconnected mode)
HiCoMo	*local committed* HiCoMo results are visible to another HiCoMo on the same MH	Optimistic timestamp ordering for a HiCoMo with base transactions	Aggregate and base tables. Convergence as correctness criteria
IOT	*local committed* transaction results are visible on the same MH	OCC globally, 2PL locally	A variation of ROWA for a fully connected environnement, optimistic evolution in disconnected mode, global certifiability as consistency criterion
Pro-motion	*local committed* transaction results are visible to local transactions on the same MH	Possibility of different isolation levels and concurrency control per compact	Compacts definiton allows several replicaton schemas
Reporting	with subtransactions *atomic, reporting and co-transactions* visibility is allowed before the commit of the global transaction		
Semantics-based	*local committed* transactions results are visible to local transactions on the same MH	2PL to control access to locally cached fragments	
Prewrite	*local committed* transactions results are visible to all hosts	2PL extended, one conflict table and new lock types are proposed	Write and prewrite data variations

Figura 19 – Sintesi sui vari aspetti dell'isolamento nei vari approcci

4.4 Durevolezza

Per rendere gli effetti delle transazioni mobili duraturi, commit locali devono essere trasformati in globali sul database server. Nel paragrafo 4.4.1 si mostra la possibilità per le transazioni che hanno eseguito il commit localmente di fare commit con successo a livello del database server. La maggior parte dei lavori

studiati non danno molto spazio alle tecniche di logging, la sezione 4.4.2 completa quest'analisi con altre proposte. La sezione 4.4.3 discute brevemente gli aspetti analizzati.

4.4.1 Garanzie di durevolezza

Clustering, *Two-tier replication*, *IOT* e *Pro-motion* non possono garantire la durevolezza prima del commit su rete fissa. *Pro-motion* con i *compact* garantisce la durevolezza, ma potrebbero esistere condizioni in cui non rispetta ciò per via delle disconnessioni; per esempio c'è una deadline (nel *compact*) che non è stata comunicata. Di conseguenza la durevolezza è difficile da ottenere nel processo di sincronizzazione. In *Reporting* le sottotransazioni sono durevoli se le transazioni genitrici hanno eseguito il commit.

HiCoMo, *Semantic-based* e Prewrite sono approcci che garantiscono la durevolezza subito dopo il commit locale.

Tuttavia il primo metodo richiede le transazioni commutative HiCoMo, oltretutto ha una complicata generazione delle *base transaction* e considera rigenerazioni e riesecuzioni delle *base transaction* (nel caso di abort). *Semantic-based* riduce la disponibilità dei frammenti perchè un MH può bloccare i frammenti per un periodo indefinito di tempo. Nell'algoritmo di *Pre-write* se una transazione mobile fa il commit locale, il commit è assicurato. L'inconveniente è lo scambio di messaggi necessario per l'ottenimento dei blocchi da parte delle BS.

Per quanto riguarda il logging, *IOT* (attualmente Coda [27]) propone un meccanismo per ridurre la grandezza dei log. In modalità disconnessa, informazioni sufficienti per ripetere gli aggiornamenti (alla riconnessione) sono mantenute in un *replay log*. Per ridurre la lunghezza del replay log, nelle operazioni di update, invece di loggare individualmente le operazioni di apertura, chiusura, scrittura, un singolo record è loggato durante l'operazione di chiusura. In letteratura si prevede anche l'esclusione di record memorizzati in precedenza per un file quando si aggiunge una nuova operazione. Ciò è possibile perchè un'operazione di memorizzazione rende tutte le precedenti versioni di un file inutili.

4.4.2 Lavori inerenti la procedura di log

Il progetto *Little Work* [28], come *IOT*, propone di ridurre la lunghezza del log. Esso propone di applicare tecniche le cui regole si basano sull'uso di un compilatore *peephole optimizers*. Sono usate delle regole per eliminare operazioni ridondanti o inutili dal log. (L'ottimizzatore ha come input un insieme di regole che permettono di passare da un set di operazioni sorgente ad un insieme di operazioni equivalenti).
L'ottimizzatore considera una lista di regole, ognuna delle quali consiste in un insieme di operazioni sorgente seguite da operazioni finali equivalenti all'insieme sorgente. Per esempio la creazione del file i seguita dalla rinomina del file i in file j potrebbe essere replicata con l'operazione di creazione del file j.
L'effetto della mobilità nel logging è analizzato in [29]. Il problema in esame è: se le transazioni mobili sono distribuite su alcuni MH, dove dovrebbero risiedere i log per garantire la durevolezza? Ci sono tre tecniche proposte:

1) L'approccio *Home BS logging* che mantiene il log del MH sulla BS su cui all'inizio era registrata (BS home) persino se il MH si muove e cambia varie BS.
Il log di una transazione mobile distribuita potrebbe essere sparso tra le home BS dei MH partecipanti.
2) L'approccio *Home MH logging* memorizza il log sulla BS che copre il MH quando la transazione è originata. Il log di una transazione è centralizzato su una BS.
3) L'approccio *Local BS logging* memorizza i log in ingresso sulle correnti BS dei MH. L'intero log di una transazione potrebbe essere sparso in varie BS.

4.4.3 Discussioni

La figura 20 mostra quando la durevolezza è assicurata e quali svantaggi ci sono. Notare che in generale, la durevolezza è garantita dopo il commit, quando i commit locali sono reintegrati dai database server. Nel caso in cui il local commit

garantisce durevolezza ,gli svantaggi sono relativi alla disponibilità dei dati ai costi di comunicazione o alla completa reintegrazione dei dati.

Per altri aspetti tecniche di ottimizzazione del logging non sono state proposte. Solo *IOT* e il progetto *Little work* affrontano il problema, sebbene servano metodi per ridurre la grandezza del log e maggiori sforzi di ricerca in materia sono necessari.

I MH non sono considerati memoria stabile e per questo si potrebbe pensare di allocare i log su FH (esempio BS). Per quanto riguarda gli approcci sulla localizzazione dei log dei MH proposti in [29] i benefici di una tecnica o di un'altra dipendono dal profilo della mobilità del MH così come dalla distribuzione dell'esecuzione della transazione mobile.

	Durability guarantees	Drawbacks
Clustering	After *commit* (reconciliation)	*Locally committed* transactions can be rolled back due to reconciliation conflicts
Two-tier replication	After *commit* (re-execution)	*Locally committed* transactions can be rolled back due to conflicts during re-execution
HiCoMo	After *local commit*	Commutative HiCoMo transactions, complicated base transaction generation, regeneration and re-executions
IOT	After *commit*	*Locally committed* transactions can be aborted due to reconciliation conflicts
Pro-motion	After *commit* (reconciliation)	*Locally committed* transactions can be rolled back due to reconciliation conflicts
Reporting	If the parent transaction *commits*, subtransactions are durable	
Semantics-based	After *local commit*	Reduction of fragments availability at database server
Prewrite	After *local commit*	Many message exchanges between MHs and BSs

Figura 20 – Sintesi sui vari aspetti della durevolezza nei vari approcci

5 Gestione del movimento e della disconnessione

I metodi analizzati in precedenza non forniscono dettagli circa la gestione della mobilità dei MH.

Solo *Pro-motion* include nella sua architettura un *mobility manager* che si occupa della comunicazione tra i MH e il database server. Si propone un'analisi complementare alla discussione dei vari approcci in letteratura.

In *KT*, *MDSTPM* e *Moflex*, le proprietà ACID non sono influenzate dalla mobilità perchè l'esecuzione delle transazioni avviene sotto la responsabilità dei DBMS localizzati su FH.

Visto che le transazioni sono richieste dai MH, la mobilità e le disconnessioni sono gestite.

Pre-serialization è un caso particolare perchè sebbene le transazioni mobili sono eseguite su rete fissa, l'atomicità e l'isolamento sono forzate permettendo le disconnessioni degli utenti mobili durante l'esecuzione delle transazioni.

5.1 Movimento e problemi di disconnessione

Con il metodo *KT* per supportare la mobilità dei MH e le disconnessioni, il *data access agent* mantiene traccia dei movimenti dei MH memorizzando una lista di tutte le BS che sono state coordinatrici per le transazioni mobili. Questa lista è usata nel caso di abort a cascata.

Ci sono anche strutture dati (*transaction status table e local log*) che memorizzano informazioni circa transazioni mobili come: ID della transazione globale, stati (attivo,commit,abort), ID delle JT, sottotransazioni che sono incluse nelle JT, transazioni di compensazione.

In *KT* due differenti modi di operare sono supportati, modo split e compensativo.

Il metodo compensativo prevede che il fallimento di alcune JT causi il disfacimento delle correnti e precedenti JT o di alcune successive .

Le JT che hanno eseguito il commit in precedenza devono essere compensate.

Questo modo di operare richiede che gli utenti provvedano alla compensazione delle transazioni e che il sistema sorgente garantisca il commit con successo.

In modalità split quando una JT fallisce non sono richieste nuove transazioni globali o locali.
Il commit e il fallimento di transazioni correntemente eseguite è una decisione lasciata al DBMS locale. Il metodo split è quello di default. Nè l'una nè l'altra modalità garantiscono la serializzabilità delle transazioni Kangaroo. Il metodo compensativo garantisce l'atomicità, ma l'isolamento potrebbe essere violato perchè il controllo della concorrenza è garantito autonomamente a livello della transazione locale.
Notare che il movimento dell'utente non tange queste proprietà.
In *MDSTPM* l'idea principale è la facilitazione delle code mediante uno scambio asincrono di messaggi, dove questi sono del tipo: *Request*, *Acknowledgment ed Information*.
Con MQF (*Message and Queuing Facility*) i MH possono sottoporre le transazioni globali e passare allo stato disconnesso.
MH e *coordinator host* mantengono delle tabelle e dei log che memorizzano lo stato complessivo dei MH e le informazioni delle transazioni globali.
(*Message Queue*, *Transactions Queue*, *Global Log*, *Global Transaction Table*, *Site Status Table*).
In qualsiasi momento un MH potrebbe richiedere informazioni circa le sue transazioni globali.
Per quanto riguarda gli aspetti di correttezza in *MDSTPM* i DBMS partecipanti sono autonomi e potrebbero essere eterogenei. Così i meccanismi di gestione delle transazioni locali (come il controllo della concorrenza) possono essere differenti e le informazioni che riguardano le esecuzioni locali (log) sono limitate o inesistenti.
Per gestire le transazioni globali, *MDSTPM* implementa il 2PL stretto e utilizza il metodo ottimistico dei ticket (OTM) per risolvere i conflitti indiretti. In *OTM* tutte le sottotransazioni globali richiedono un ticket. Questo causa conflitti addizionali tra sottotransazioni e l'ordine di esecuzione è determinato dai ticket.

In *Moflex* due caratteristiche riguardanti la mobilità sono messe in luce:

- l'esecuzione di transazioni dipendenti dalla localizzazione[3];
- l'influenza della disconnessione nell'esecuzione delle transazioni.

Nella definizione delle transazioni, gli utenti possono specificare se una sottotransazione è dipendente dalla localizzazione o meno. Per le sottotransazioni dipendenti dalla localizzazione, devono essere specificate delle regole di controllo dell' *handoff*. Le scelte sono:

1) continuare l'esecuzione della transazione sulla nuova cella (***continue***);
2) abortire la transazione della cella precedente e reiniziarla sulla cella nuova (***restart***);
3) eseguire il commit delle operazioni eseguite sulla vecchia cella e le rimanenti operazioni eseguirle sulla nuova cella (***split-resume***);
4) le operazioni eseguite sulla vecchia cella fanno commit e la transazione è eseguita interamente sulla nuova cella (***split-restart***).

Per le operazioni *split* si utilizza qui un metodo simile a quello usato nelle *KT*. Quando un MH salta da una BS ad un'altra, le transazioni sono divise ed il coordinatore è riallocato sulla nuova BS.

Le definizioni delle transazioni potrebbero anche includere stati *goals* che indicano stati finali accettabili. Si utilizza il protocollo di 2PC. Il gestore delle transazioni mobili della cella ,dove una delle sottotransazioni raggiunge uno stato traguardo accettabile, diviene il coordinatore del protocollo di 2PC per il commit globale.

Nel caso di *Pre-serialization* il principio è garantire atomicità e isolamento (A/I), supportando disconnessioni e migrazioni degli utenti mobili durante l'esecuzione delle transazioni.

Le transazioni su sito di una globale sono organizzate in vitali e non vitali. Le proprietà A/I sono rispettate solo da un insieme di transazioni su sito vitali. L'abort di transazioni su sito non vitali non forza l'abort della transazione globale. Il tempo tra la sottomissione della prima transazione vitale su sito e il completamento dell'ultima è chiamato fase vitale della transazione globale. Le transazioni globali potrebbero essere in uno degli stati seguenti:

1) attiva, l'utente è connesso e l'esecuzione continua;
2) disconnessa, l'utente è disconnesso ma la disconnessione era prevista e la riconnessione è attesa per cui l'esecuzione continua;
3) sospesa, l'utente è disconnesso e si crede che abbia avuto un guasto catastrofico;

4) committed o aborted, la transazione ha fatto il commit o l'abort.

L'esecuzione della transazione non si ferma quando l'utente è disconnesso (in modo previsto).

Tutte le risposte inviate ad un utente disconnesso sono inviate alla riconnessione.

Il controllo delle transazioni globali migra da un *global coordinator* ad un altro in accordo al movimento dell'utente.

Il *site transaction manager* soprassiede all'esecuzione delle sue transazioni su sito. Queste (quelle vitali e non) potrebbero essere in uno dei quattro stati:

1) *active*;
2) *completed* (la transazione su sito fa il commit sul database locale ma la transazione globale non ha fatto il commit);
3) *committed* (la transazione su sito e la rispettiva transazione globale hanno fatto il commit);
4) *aborted*.

Se la transazione globale è nello stato disconnesso, l'esecuzione delle transazioni su sito continua. Un *global coordinator* memorizza i messaggi per gli utenti disconnessi, li invia ad essi alla riconnessione e riattiva le transazioni disconnesse.

Il coordinatore globale verifica le proprietà di atomicità e isolamento eseguendo l'algoritmo del grafo parziale della serializzabilità globale (PGSG) alla fine della fase vitale.

Se le proprietà di A/I sono violate, la transazione globale è abortita; altrimenti è messa nello stato di toggle.

Dopo questo stato una transazione globale mobile potrebbe iniziare solo transazioni non vitali sul sito. Una volta che lo stato è toggle la transazione globale stabilisce il suo ordine di serializzabilità nello schema di serializzazione globale e per essa è garantito il commit.

Alla fine dell'esecuzione ogni transazione nello stato toggle fa girare per la seconda volta l'algoritmo PGSG.

L'obiettivo è verificare che transazioni su sito non vitali non violino l'ordine di serializzabilità stabilito quando la transazione è passata in toggle. Ogni transazione non vitale che viola questo ordine è abortita senza inficiare la transazione globale.

Una transazione globale nello stato toggle è abortita solo se prevede l'esecuzione di un'altra transazione globale mentre è nello stato sospeso.

Così *Pre-serialization* garantisce atomicità semantica e serializzabilità globale. Una versione modificata di *Pre-serialization* è proposta e le proprietà di atomicità e isolamento sono forzate solo per le transazioni vitali. In questa versione l'algoritmo PGSG è eseguito una sola volta alla fine della fase vitale.

5.2 Confronti tra i vari modelli in letteratura

KT e *MDSTPM* sono molto simili.
Essi propongono di aggiungere uno strato nelle architetture esistenti multidatabase per manipolare le transazioni richieste dai MH.
La principale differenza è la scelta del coordinator host.
In *MDSTPM* il coordinatore dell'esecuzione delle transazioni mobili è centralizzato. Il coordinatore FH è fissato in anticipo e non cambia durante l'intera esecuzione.
In *KT* la coordinazione è distribuita tra tutte le BS visitate dai MH.
Perciò notiamo che le KT trattano la natura mobile dei MH, non solo le disconnessioni. La coordinazione distribuita riduce i costi di comunicazione durante l'esecuzione; comunque nel caso di abort a cascata i costi di comunicazione crescono notevolmente.
Con il coordinatore centralizzato di *MDSTPM* gli abort a cascata sono semplici ed economici; comunque nel caso di alta mobilità le comunicazioni potrebbero essere costose.
Tre possibili approcci per la coordinazione delle transazioni sono analizzati:
1) localizzazione della coordinazione sul MH;
2) localizzazione su un FH centralizzato;
3) migrante di BS in BS.
Riguardo altri aspetti si propone una definizione delle transazioni mobili dedicata alla dipendenza dalla localizzazione dei dati [29].
Si analizza l'impatto della mobilità su tali dati e il loro effetto sulle proprietà acid.
Nell'elaborazione mobile l'adattabilità alle variazioni dell'ambiente è un problema importante.
Inoltre in relazione all'adattabilità degli *handoff*, tra i lavori analizzati, solo Moflex è interessata alla localizzazione dei dati.

Lo svantaggio principale di *Moflex* è che gli utenti devono provvedere alla complicata definizione delle transazioni. Sono ancora affrontati in merito problemi relativi all'adattabilità [30,31].
L' eccezionalità di *Pre-serialization* è che le proprietà di atomicità e isolamento sono adatte ai contesti multi-database che considerano le disconnessioni degli utenti mobili.
Tuttavia l'algoritmo PGSG è costoso per via del principio di propagazione (diffusione della serializzabilità delle informazioni) ed anche perchè potrebbe essere eseguito due volte durante l'elaborazione delle transazioni globali (nella versione base).

6 Conclusioni e direzioni di ricerca

Si sono esplorati i metodi per la gestione delle transazioni mobili.
Questo argomento è particolarmente importante oggi che i sistemi informativi coinvolgono sistemi mobili e host fissi raggiunti attraverso una combinazione di reti wireless e fisse.
Negli ultimi anni molti sforzi accademici ed industriali sono stati dedicati a migliorare la gestione dei dati negli ambienti mobili.
Diversamente dai tradizionali ambienti centralizzati e distribuiti, gli ambienti mobili sono altamente versatili e sfruttano risorse limitate. Come conseguenza l'esecuzione delle transazioni mobili non è prevedibile e richiede approcci adattati.
Ci sono alcune ragioni che conducono allo sviluppo di vari approcci nuovi per le transazioni mobili. Una profonda analisi dei progetti di ricerca guida alla classificazione in due gruppi.
Il primo rilassa le proprietà acid per permettere l'esecuzione delle transazioni su MH. Metodologie in questo gruppo di solito ignorano il movimento dei MH.
Il secondo gruppo delle proposte riguarda la richiesta delle transazioni da parte dei MH e l'esecuzione sui FH. In questo caso il movimento dei MH è considerato durante l'esecuzione della transazione.
Questi studi mostrano che le proprietà acid sono difficili da applicare negli ambienti mobili.

Le ragioni principali sono l'autonomia necessaria per lavorare in modalità disconnessa o connessa debolmente e l'inerente mobilità dei MH. Quasi tutti i progetti analizzati realizzano le proprietà acid per migliorare la flessibilità e reggere i vincoli degli ambienti mobili.

Modelli avanzati delle transazioni sono chiaramente appropriati per gli ambienti mobili. A dispetto del gran numero di lavori sulle transazioni mobili, alcuni problemi di ricerca rimangono aperti, tra essi, il logging e la gestione adattabile delle transazioni mobili.

La tecnica per il log è stata rivisitata per affrontare le limitate capacità di memoria dei MH. La compressione del log e l'ottimizzazione degli accessi sono quasi ignorati dagli approcci analizzati.

Anche i MH potrebbero essere persi ed esposti ad incidenti. Le strategie di trasferire i log locali sui FH (BS o server sono considerati memoria stabile) servono per garantire la durevolezza delle transazioni che hanno eseguito il commit localmente.

Un altro importante punto è l'adattabilità per via delle variazioni dell'ambiente mobile (localizzazione, ampiezza di banda, ambiente, distanza dal server) o delle limitazioni delle risorse di calcolo mobili (memoria/ potenza).

Quasi tutti i sistemi studiati adattano il loro comportamento per supportare il modo connesso e non. Solo pochi metodi considerano l'adattabilità relativa agli spostamenti dei MH (es hand-off). Comunque le variazioni non sono limitate alle disconnessioni e hand-off.

Una scelta dinamica dei modelli di esecuzione delle transazioni dipende dalla localizzazione corrente, risorse dei MH o rete fissa, e fa crescere certamente il tasso di successo delle transazioni.

CAPITOLO 3

Un modello teorico per la gestione della concorrenza di "long lived transatcion"

Qui di seguito verrà descritto il modello teorico alla base del protocollo proposto per la gestione della concorrenza di Long-Lived Transaction in ambiente mobile.
Obiettivi di questo protocollo sono l'incremento della concorrenza e la gestione della disconnessione.
L'idea di base è quella di modificare il 2PL classico per tenere conto delle frequenti disconnessioni dei dispositivi che hanno generato le transazioni e allo stesso tempo di considerare quella che è la compatibilità semantica delle operazioni che costituiscono le transazioni per aumentare l'accesso in concorrenza alle risorse.
Il tradizionale approccio 2PL risulta infatti inapplicabile perchè un'imprevedibile disconnessione può causare il blocco di una risorsa per un tempo indefinito salvo che non si forzi un abort a disconnessione avvenuta.
L'abort preventivo di transazioni disconnesse, però comporterebbe un elevato numero d'abort inutili e, visto che in ambiente mobile la disconnessione si può vedere come un evento fisiologico e non patologico, è evidente che tale tecnica è inapplicabile.

1 Definizioni preliminari

Qui di seguito sono fornite le definizioni su cui si poggia il modello teorico del protocollo proposto per la gestione della concorrenza per *Long-Lived Transaction* in ambiente mobile.

1.1 Mobile Transaction

Definiamo *Mobile Transaction* una generica transazione generata da un dispositivo mobile, che, però può coinvolgere per la sua esecuzione anche uno o più dispositivi dislocati su rete fissa.
La comunicazione tra dispositivi mobili e fissi avviene mediante le BS.

1.2 Compatibilità tra operazioni

Detta $op_i^j(X)$, la i-esima operazione sul dato X della transazione j-esima sul dato X ,definiamo:

- $Xread^j$ il valore del dato X letto su DB dalla j-esima transazione prima di modifiche apportate su esso da parte di una generica altra operazione $op_i^j(X)$
- Xt^j replica di X sulla quale opereranno le $op_i^j(X)$ prima dell'esecuzione del Commit da parte della MT_j
- $Xold^j$ il valore del dato X presente su database all'atto della richiesta di commit da parte della j-esima transazione prima dell'esecuzione dell'algoritmo di riconciliazione
- $Xnew^j$ il nuovo valore del dato X da scrivere su database dopo la richiesta di commit della transazione j-esima e l'esecuzione dell'algoritmo di riconciliazione

N operazioni $\{op_i^j(X),\ldots,op_k^h(X)\}$ si dicono compatibili se esiste un algoritmo di riconciliazione in grado di determinare, a partire dai rispettivi *Xread Xt Xold* e

dalla semantica delle operazioni coinvolte, il valore corretto del dato *Xnew* qualunque sia l'ordine di esecuzione delle operazioni.

Da aggiungere anche che non è, in tale modello, considerato un problema la lettura inconsistente del dato, ovvero il fatto che una transazione legge un dato e vi opera nonostante nel frattempo tale dato possa essere stato modificato da un'altra transazione.

1.3 Conflitto tra Mobile Transaction

Due generiche transazioni mobili MT_i e MT_j si dicono in conflitto su una data risorsa *X*, se una delle due detiene il lock su *X*, e l'altra in concorrenza con essa attua una richiesta di lock sulla medesima risorsa.

1.4 Compatibilità tra operazioni generate da Mobile Transaction

Siano MT_i e MT_j due generiche transazioni mobili in conflitto su una data risorsa *X*, e siano $Op_i(X)$ e $Op_j(X)$ i due insiemi di operazioni elementari generate dalle transazioni *MTi e MTj* sulla risorsa *X*.
Si dice che le operazioni generate dalle due mobile transaction sono compatibili se e solo se le operazioni dell'insieme $Op = Op_i(X) \cup Op_j(X)$ sono compatibili.

1.5 Stato di una Mobile Transaction

Lo stato di una Mobile Transaction è una condizione in cui una Mobile Transaction può trovarsi durante il suo ciclo di vita.
L'insieme degli stati ammissibili per una Mobile Transaction sono i seguenti:

$$\boldsymbol{S_{MT}} = \{Run, Wait, Disconnected, Commit, Abort\}$$

- *Run (Active):* La MT si trova in tale stato quando la Mobile unit che l'ha generata risulta connessa e non è in coda su nessuna risorsa.

- *Wait:* La MT si trova in tale stato quando è in coda su una risorsa.
- *Disconnected:* La MT si trova in tale stato quando la Mobile Unit che l'ha generata è rimasta inattiva oltre un determinato timeout a causa di mancanza di rete o negligenza dell'utente.
- *Commit:* La MT si trova in tale stato quando ha portato a termine correttamente la sua esecuzione.
- *Abort:* La MT si trova in questo stato quando la sua esecuzione è terminata in maniera non corretta e nessuna modifica da lei apportata è stata resa effettiva su database.
- *BeginMT:* Non è un vero e proprio stato ma indica ciò che da inzio ad una MT.

1.6 Stato di una Risorsa

Lo stato di una Risorsa è una condizione in cui una Risorsa può trovarsi durante il suo ciclo di vita.

L'insieme degli stati ammissibili per una Risorsa sono i seguenti:

$$\boldsymbol{S_R}=\{Free,Busy,Busy\_Disc\}$$

- *Free*: La Risorsa non è detenuta da nessuna transazione.
- *Busy*:La risorsa è detenuta da una o più transazioni, e almeno una di queste non si trova nello stato *Disconnected* .
- *Busy_Disc*: La risorsa è detenuta da una o più transazioni il cui stato è *Disconnected.*

Per caratterizzare una risorsa *X*, oltre allo stato $s_r \in \boldsymbol{S_R}$, dobbiamo anche specificare l'insieme di operazioni *Op* appartenenti alle generiche MT_k che stanno agendo su di essa.

Quindi utilizziamo la tripla

$$<X, s_r, Op>$$

1.7 Transizione di Stato

Per transizione di stato intendiamo il passaggio da uno stato all'altro di una Mobile Transaction.

Possiamo definire la funzione di transizione di stato nel modo seguente:

$$\delta : S_I \subset S_{MT} \times I \rightarrow S_{MT}$$

Dove:

$$\boldsymbol{S_{MT}}=\{Run, Wait, Disconnected, Commit, Abort\}$$

Per quanto concerne gli input dobbiamo distinguerli in due sottoinsiemi:

Input sincroni

$$I1 = \left\{\mathrm{Re}\, q\_Lock(op_i^j, < X, \mathrm{s_r}, Op >), \mathrm{Re}\, q\_Com, \mathrm{Re}\, q\_Abo\right\}$$

Input Asincroni

$$I2 = \left\{Timeout\_W, Timeout\_D, Unlock, Disconnection, \mathrm{Re}\, connection, Abort\_Forced\right\}$$

$$I = I1 \cup I2$$

- $\mathrm{Re}\, q\_Lock(op_i^j, < X, \mathrm{s_r}, Op >)$: la transazione MT_j effettua una richiesta di lock su una determinata risorsa $< X, \mathrm{s_r}, Op >$ specificando l'operazione op_i^j da effettuare su quest'ultima.
- $\mathrm{Re}\, q\_Com$: la transazione richiede il commit del suo operato.
- $\mathrm{Re}\, q\_Abo$: la transazione effettua l'abort del suo operato.
- $Timeout\_W$: input asincrono dovuto alla permanenza per un tempo superiore al Timeout nello stato di Wait (nel caso di risoluzione di deadlock mediante il "wait for graph" tale input viene sostituito da *Deadlock_Recognition*)
- $Timeout\_D$:input asincrono dovuto alla permanenza per un tempo superiore al Timeout nello stato di Disconnected
- *Unlock* : input asincrono dovuto alla liberazione della risorsa in precedenza trovata occupata dalla transazione.
- *Reconncetion*: input asincrono segnalante l'avvenuta riconnessione di una transazione disconnessa.
- $Abort\_Forced$: input asincrono dovuto alla richiesta di lock sulla risorsa X detenuta da $\mathrm{MT_j}$ da parte di una transazione $\mathrm{MT_k}$ $/ S^j := Disconnected$; $op_i^j(\mathrm{X})$, $op_h^k(\mathrm{X})$ incompatibili.

2 Il modello proposto

Sulla base delle definizioni date nel paragrafo precedente risulta possibile formalizzare il ciclo di vita di una Mobile Transaction all'interno del nostro sistema attraverso il seguente diagramma degli stati.

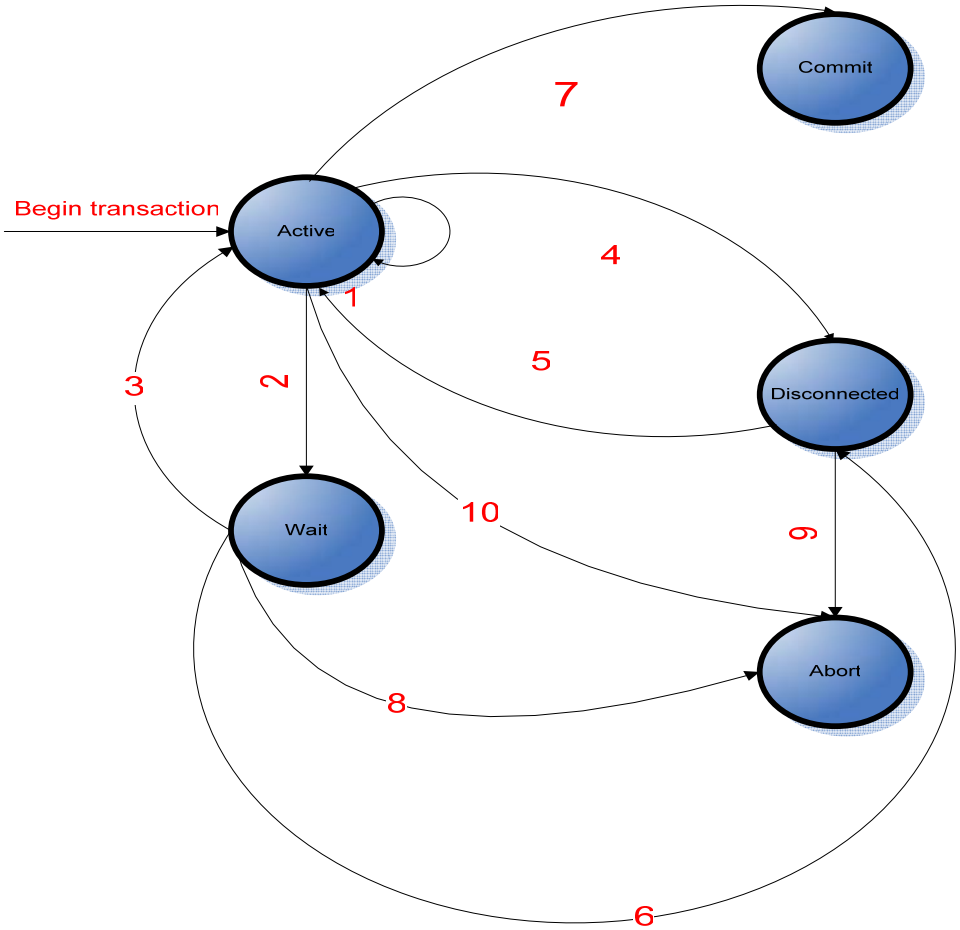


Figura 1 – Diagramma degli stati

Per comprendere al meglio la modalità di esecuzione del nostro protocollo riteniamo utile descrivere in maniera dettagliata come avvengono le possibili transizioni di stato di una transazione.

	S	I	S	Ramo
1	*Run*	$\mathrm{Re}\,q\_Lock(op_i^j, < X, Free, \_ >)$	*Run*	1
2	*Run*	$\mathrm{Re}\,q\_Lock(op_i^j, < X, Busy, Op >)$ / op_i^j ,*Op Compatibili*	*Run*	1
3	*Run*	$\mathrm{Re}\,q\_Lock(op_i^j, < X, Busy\_D, Op >)$	*Run*	1
4	*Run*	$\mathrm{Re}\,q\_Lock(op_i^j, < X, Busy, Op >)$ op_i^j ,Op *InCompatibili*	*Wait*	2
5	*Run*	$\mathrm{Re}\,q\_Com$	*Commit*	7
6	*Run*	$\mathrm{Re}\,q\_Com$	*Abort*	10
7	*Run*	$\mathrm{Re}\,q\_Abo$	*Abort*	10
8	*Run*	*Disconnection*	*Disconnected*	4
9	*Wait*	*Unlock*	*Run*	3
10	*Wait*	*Timeout_W*	*Abort*	8
11	*Wait*	*Disconnection*	*Disconnected*	6
12	*Disconnected*	Re*connection*	*Run*	5
13	*Disconnected*	*Timeout_D*	*Abort*	9
14	*Disconnected*	*Abort_Forced*	*Abort*	9

Figura 2 – Tabella delle transizioni di stato

2.1 Algoritmo 1,2

*C*aso in cui la transazione j-esima richiede il lock,per l'esecuzione di op_i^j *(X)*, su

di una risorsa libera o lockata da transazioni le cui operazioni su tale risorsa sono compatibili con $op_i^{\ j}(X)$.

```
Begin
        <creo X^j_read , X^j_t >;
        X^j_read := X ;
        X^j_t := X ;
        S^j := Run ;
End .
```

2.2 Algoritmo 3

Caso in cui la transazione j-esima richiede il lock,per l'esecuzione di $op_i^{\ j}(X)$, su di una risorsa lockata da transazioni disconnesse le cui operazioni *Op* su tale risorsa sono incompatibili con $op_i^{\ j}$.

```
Begin
        <creo X^j_read , X^j_t >;
        X^j_read := X ;
        X^j_t := X ;
        S^j := Run ;
// vengono abortite tutte le transazioni MT1…MTw disconnesse che detenevano il
// lock sulla risorsa R e le cui operazioni risultano incompatibili con op_i^j
for k:=1 to w do
    S^k := Abort ;
End ;
```

2.3 Algoritmo 4

Caso in cui la transazione MT_j richiede il lock su X ,per l'esecuzione di op_i^j (X), essendo X lockata da transazioni MT_k le cui operazioni Op(X) su tale risorsa sono incompatibili con op_i^j (X).

```
Begin
// in tal caso la transazione MT_j viene posta in coda
//  sulla risorsa e quindi il suo stato passa a Wait
        S^j := Wait ;
End;
```

2.4 Algoritmo 5,6

Caso in cui la transazione MT_j richiede il commit.

```
Begin
        for k:=1 to num_risorse  do
        begin
                <creazione  X^j_old  e inizializzazione al valore presente su DB>;
                <creazione  X^j_new >;
                 X^j_new :=Algoritmo_Riconciliazione( X^j_read , X^j_t , X^j_old );
        end;
        <generazione transazione su DB per aggiornare i dati mediante { X^j_new } >
        if (Commit_Transaction ) then
         S^j := Commit;
        Else
         S^j := Abort;
for k:=1 to num_ris  do
        <deallocazione  X^j_old , X^j_new , X^j_read , X^j_t  >
End.
```

2.5 Algoritmo 7

Caso in cui la transazione MT_j richiede l'abort.

Begin
$S^j := Abort$;
for k:=1 to num_ris do
$<dealloca\ X^j_{read}, X^j_t >$;
End;

2.6 Algoritmo 8,11

Caso in cui si verifica un evento di disconnessione.

Begin
$S^j := Disconnected$;
End;

2.7 Algoritmo 9

Caso in cui la risorsa sulla quale la MT_j stava in coda per eseguire l'operazione op_i^j diventa Free

Begin
$< crea\_X^j_{read}, X^j_t >$;
$X^j_{read} := R$;
$X^j_t := R$;
$S^j := Run$;
End.

2.8 Algoritmo 10,13,14

10) caso in cui MT_j permane nello stato di Wait per un tempo maggiore di $Timeout\_W$;

13) caso in cui MT_j permane nello stato di *Disconnected* per un tempo maggiore di $Timeout\_D$;

14) caso in cui MT_j detine il lock su almeno una risorsa X richiesta da MT_k per l'esecuzione di op_k^h (X) ed: $S^j := Disconnected$; op_i^j (X), op_h^k (X) incompatibili.

```
Begin
        for k:=1 to num_ris  do
        <dealloca  X^j_read , X^j_ti >;
        S^j := Abort;
End;
```

2.9 Algoritmo 12

Caso in cui MT_j effettua una riconnessione

```
Begin
        S^j := Run;
End;
```

3 Sintesi sui modi di operare nel modello proposto

Cerchiamo di illustrare in breve ciò che avviene utilizzando l'approccio proposto.
Una MT nel momento in cui viene istanziata viene a trovarsi nello stato di *Run*.
In seguito ad una richiesta di lock può permanere in tale stato o migrare nello stato di *Wait* a seconda di vari fattori:

- Stato della risorsa
- Operazione che intende effettuare sulla risorsa
- Operazioni effettuate sulla risorsa dalle MT che la detengono
- Stato delle MT

Nel caso di risorsa *X Free*,lo stato di quest'ultima passa a *Busy* mentre la *MT* rimarrà in *Run*.
La transazione memorizza il valore del dato presente su DB in *Xread* e crea il dato di appoggio *Xt.*
Le modifiche apportate dalle operazioni di cui è costituita la MT avranno effetto esclusivamente su *Xt*.
Nel caso di risorsa X *Busy*,viene effettuato un controllo tra l'operazione che ci accingiamo ad effettuare e quelle che vi stanno operando appartenenti a MT che già detengono il lock su X.
Se il controllo definisce compatibili le operazioni nulla cambia rispetto al caso visto in precedenza.
In caso contrario la MT passerà nello stato di Wait e vi permarrà fintanto che la risorsa non venga liberata a meno che il tempo di permanenza in tale stato non risulti superiore ad un fissato *Timeout*, evento che scatenerebbe l'Abort della MT.
L'ultimo stato in cui può trovarsi una generica risorsa *X* è *Busy_D*,ovvero tutte le MT che la detengono risultano disconnesse.
Questo stato viene introdotto per evitare Abort inutili di MT disconnesse.
L'introduzione di questo stato permette infatti a MT disconnesse di poter continuare ad operare a riconnessione avvenuta come se nulla fosse accaduto a meno che le risorse da esse detenute non blocchino l'esecuzione di altre MT.

Una MT disconnessa viene mandata in abort se e solo se una transazione in run dovesse richiedere un lock su risorse detenute da essa per eseguire operazioni incompatibili.

All'atto del Commit da parte della MT verrà eseguito per ogni dato d'appoggio un algoritmo di riconciliazione in modo tale che il generico *Xt* contenga il valore effettivo da riportare su DB.
Eseguiti tutti gli algoritmi di riconciliazione necessari viene generata una transazione su DB che effettua il matching dei dati reali con quelli di appoggio.
Se tale transazione dovesse andare a buon fine la MT andrà in Commit,in caso contrario sarà Abortita.
Qui di seguito mostriamo qualche esempio di operazioni compatibili e incompatibili.
Per quanto concerne le operazioni {+,-,*,/,=} possiamo subito affermare che:

- Le operazioni di addizione (+) e sottrazione (-) che coinvolgono operandi costanti sono compatibili
- Le operazioni di moltiplicazione(*) e divisione(/) che coinvolgono operandi costanti sono compatibili
- L'assegnazione (=) non è compatibile con nessun tipo di operazione.

L'algoritmo di riconciliazione per le operazioni di addizione e sottrazione e il seguente:

$$Xnew = Xt + Xold - Xread$$

L'algoritmo di riconciliazione per le operazioni di moltiplicazione e divisione è il seguente:

$$Xnew = \frac{Xt}{Xread} \times Xold$$

Ovviamente l'operazione di assegnazione non è compatibile con alcun tipo di operazioni in quanto il risultato finale è dipendente dall'ultima operazione effettuata,infatti il valore del dato è quello settato dall'ultima operazione effettuata.

4 Problematiche e relative soluzioni

In questo paragrafo consideriamo come nell'approccio si affrontano i problemi di starvation, deadlock, percentuale di abort e serializzabilità.

4.1 Starvation

L'introduzione del concetto di compatibilità tra operazioni potrebbe indurre una MT ad attendere un considerevole intervallo di tempo prima che la risorsa le venga concessa.
Addirittura la MT potrebbe trovarsi nello stato di attesa illimitata e quindi in *starvation*.
Infatti, anche se non si è introdotto il concetto di *priorità,* è come se MT che richiedono il lock su di una risorsa *<X,Busy,Op>* per eseguire operazioni compatibili con quelle dell'insieme Op avessero priorità maggiore rispetto ad una MT che invece è intenzionata ad eseguire un'operazione incompatibile.
Per evitare questo fenomeno si è scelto di mettere in coda anche transazioni intenzionate ad effettuare un'operazione *op* su *X* compatibile con le operazioni *Op* già operanti sulla medesima risorsa, se su tale risorsa vi dovessero essere transazioni incompatibili in coda.

4.2 Deadlock

Ricordiamo che il deadlock è una condizione in cui due transazioni MT1 e MT2 si trovano ad occupare due risorse A e B e ciascuna chiede il lock esclusivo sulla risorsa occupata dall'altra. Ciò provoca un blocco delle due MT, che non possono procedere,quindi nel nostro sistema rimarrebbero perennemente in stato di wait. La probabilità di deadlock cresce in modo lineare con il numero di transazioni e in modo quadratico con il numero di richieste di lock da parte di ogni transazione.

Teorema

Il protocollo proposto per la gestione della concorrenza non introduce ulteriori cause di deadlock rispetto al 2PL tradizionale.

Dimostrazione

- *Worst case:* le MT in conflitto sulle risorse effettuano operazioni incompatibili.
    - Approccio proposto ⇔2PL

- le MT in conflitto sulle risorse effettuano operazioni compatibili.
    - Condizione necessaria per il verificarsi di un deadlock è la richiesta da parte delle MT di un lock esclusivo sulla risorsa occupata dall'altra.In questa caso il lock sulle risorse per le MT è un lock condiviso.
- Introduzione stato Disconnected per le MT
    - L'introduzione di tale stato non è causa di ulteriori deadlock in quanto una MT disconnessa perde il lock sulla risorsa non appena dovesse giungere una richiesta di lock da parte di una MT che intenda effettuare operazioni incompatibili.

Per quanto detto quindi nel protocollo presentato il deadlock può essere gestito come in un classico protocollo di 2PL.
Alcune delle soluzioni possibili sono le seguenti:

- *Uso del Timeout*
    - Viene fissato un tempo *t* di timeout oltre il quale le transazioni in attesa di lock vengono annullate (uccise)
    - Vantaggi
        - Molto semplice
    - Svantaggi
        - *t* alto risolve tardi il problema
        - *t* basso uccide troppe transazioni

- Riconoscimento dello stallo
    - Costruzione incrementale del grafo di attesa (*wait-for graph*): i nodi sono le transazioni e l'arco da Ti a Tj in questo grafo significa che Ti sta aspettando che Tj liberi una risorsa

• Quando si genera un ciclo, lo schedule risolve lo stallo scegliendo la transazione vittima (ad esempio quella che effettuato meno lavoro – ma attenzione al blocco individuale, cioè il rischio di uccidere ripetutamente la stessa transazione) ed eseguendo il suo rollback (con tutte le conseguenze del caso)

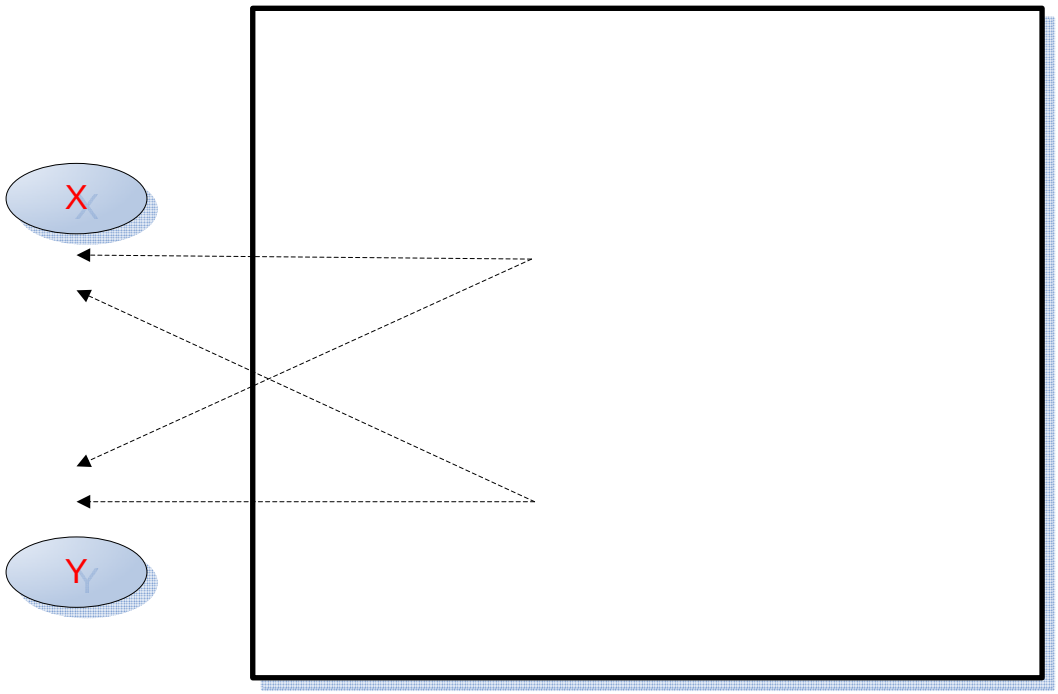


Figura 3 – Wait for graph (riconoscimento dello stallo)

4.3 Abort Eccessivi

L'esecuzione concorrente di più operazioni, nonostante la compatibilità delle stesse,potrebbe portare all'atto della riconciliazione tra i dati reali X e le repliche Xt, a numerosi Abort di MT a causa di violazione di vincoli su DB.
Un esempio renderà più chiaro quanto detto.
Sia X=10 il numero di biglietti disponibili per un concerto.
La prenotazione di tali biglietti, si può vedere come un'operazione compatibile con altre prenotazioni.

Infatti l'operazione di prenotazione non è nient'altro che un'operazione di sottrazione

$$X = X - K$$

dove:

K=numero biglietti prenotati

Siano :

Ki=Numero biglietti prenotati dalla MTi

t_i^a *=Istante in cui ha inizio la MTi*

t_i^c *=Istante in cui la MTi richiede il commit*

In una situazione del genere l'esito di una MT dipenderà dal momento in cui verrà effettuata la richiesta di commit.

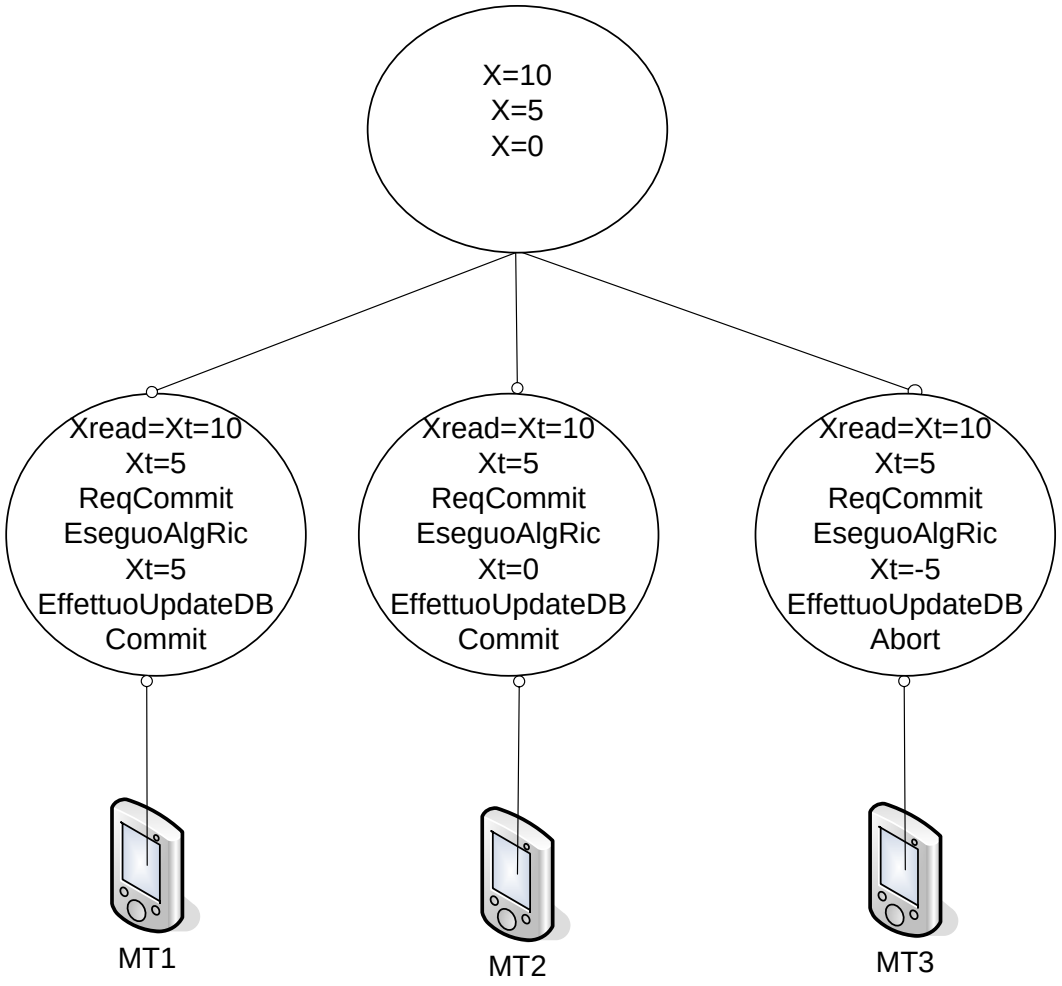


Figura 4 – Esempio relativo alla gestione della percentuale di abort

Con:

$t_1^a < t_2^a < t_3^a < t_1^c < t_2^c < t_3^c$;

K1= K2= K3=5;

Nel nostro protocollo abbiamo previsto, per tutte le operazioni per le quali sia possibile definire uno scostamento medio Δ del valore del dato X apportato dalla singola operazione,un numero $N_{\max}$ massimo di operazioni in concorrenza su X . Ad esempio per l'esempio su esposto

$$N_{\max} = \frac{X}{\Delta}$$

Quindi il numero massimo di operazioni in concorrenza su un dato X è funzione del valore del dato e dalla modifica media apportata su quest'ultimo dalla generica operazione.

4.4 Serializzabilità

La serializzabilità del metodo proposto è garantita dalla seguenti condizioni:

1. le operazioni compatibili che vanno in concorrenza sulla stessa risorsa operano su copie virtuali di quest'ultima, e, quindi durante la loro esecuzione non hanno possibilità di cambiare i valori dei dati presenti sul database;
2. il commit "effettivo" delle transazioni sul database viene sempre gestito attraverso un protocollo 2PL classico che è di per sé serializzabile;
3. L'ordine con cui sono eseguite le operazioni compatibili in concorrenza su una data risorsa non influenza il valore finale del dato sul database.

In altri termini le modifiche apportate al 2Pl classico non alterano la sua serializzabilità.

CAPITOLO 4

Progettazione ed Implementazione di uno scheduler transazionale per la gestione delle transazioni in ambiente mobile

Obiettivo del presente capitolo è quello di illustrare l'infrastruttura software di supporto al sistema per la gestione della concorrenza delle transazioni in ambiente mobile.
Verrà di seguito presentata la progettazione e l'implementazione del prototipo dello scheduler transazionale.

1 Introduzione

Dal punto di vista dell'end-user, il sistema offre un insieme di servizi per la gestione delle transazioni in ambiente mobile, erogati secondo il paradigma di comunicazione client-server. Da un punto di vista strutturale il sistema può, invece, essere concepito come una classica architettura multi-layer, basata sui seguenti livelli:

- **LDBS Layer (Back-End):** a questo livello si collocano i dati di interesse per le applicazioni.
- **GTM Layer**: a questo livello si collocano i servizi offerti e il middleware che svolge le funzioni di scheduler transazionale.
- **Client o Presentation Layer (Front-End):** a questo livello si collocano le interfacce ed i programmi utente, attraverso i quali è stato possibile testare il sistema.

Il flusso dei dati è originato dal Client Layer,che richiede un servizio, ed è diretto verso il GTM Layer che, da una lato, instaura una comunicazione con il LDBS

Layer al fine di reperire i dati necessari a rispondere alla richiesta e, dall'altro effettua, la serializzazione delle transazioni in concorrenza su una stessa risorsa. Il flusso dei dati termina dunque con la risposta che il GTM Layer restituisce al Client Layer.

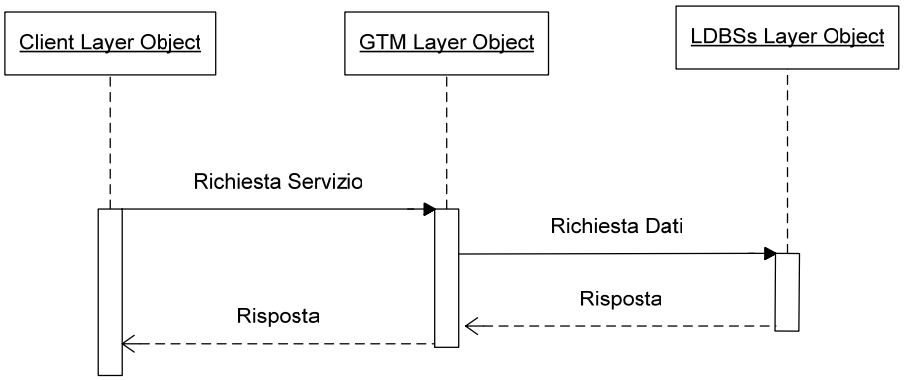


Figura 1 – Flusso di comunicazione per l'architettura proposta

La figura 1 illustra il suddetto flusso di comunicazione.

Nel seguito è descritta la soluzione proposta, sia hardware che software, per l'implementazione della suddetta infrastruttura.

2 Architettura hardware di supporto

Da un punto di vista hardware, si prevede una configurazione minimale composta dalla macchina server, da una base station e da un dispositivo portatile. Più nel dettaglio si individuano le seguenti componenti architetturali:

- **GTM Server**: da un lato funge da interfaccia di front-end per le richieste utente, dall'altro, permette il processing delle applicazioni client, occupandosi, nel contempo, delle problematiche di sicurezza e concorrenza ad esse relative (autenticazione utenti, connessione al database, schedulazione, etc…).
- **LDBS Server**: ha le funzionalità di gestione e memorizzazione dei dati tipiche di un classico Database Management System; al suo interno si trovano i "database schema" relativi a tutti i dati a cui le applicazioni possono accedere.
- **Base Station**: l'entità funzionale delegata a rendere possibile la comunicazione tra Mobile Unit e GTM Server in una particolare area geografica;
- **Mobile Unit**: dispositivo portatile sul quale viene mandata in esecuzione l'applicazione client sviluppata.

L'architettura proposta rispecchia un approccio di tipo thin-client, in cui la complessità computazionale è a carico del solo GTM layer. Chiaramente il collo di bottiglia del sistema è costituito dal GTM, che potrebbe, a tale proposito essere "deployato" su più macchine server in un'ottica di "grid computing". I dispositivi mobili in questo scenario si limitano ad inviare semplici richieste e a visualizzare i relativi risultati, richiedendo al server l'esecuzione delle operazioni.

Questa architettura è particolarmente adatta per terminali dalle limitate risorse computazionali e di storage.
In figura 2 è illustrata una possibile architettura hardware di supporto che tiene conto delle specifiche sopra elencate e dello scenario di riferimento.

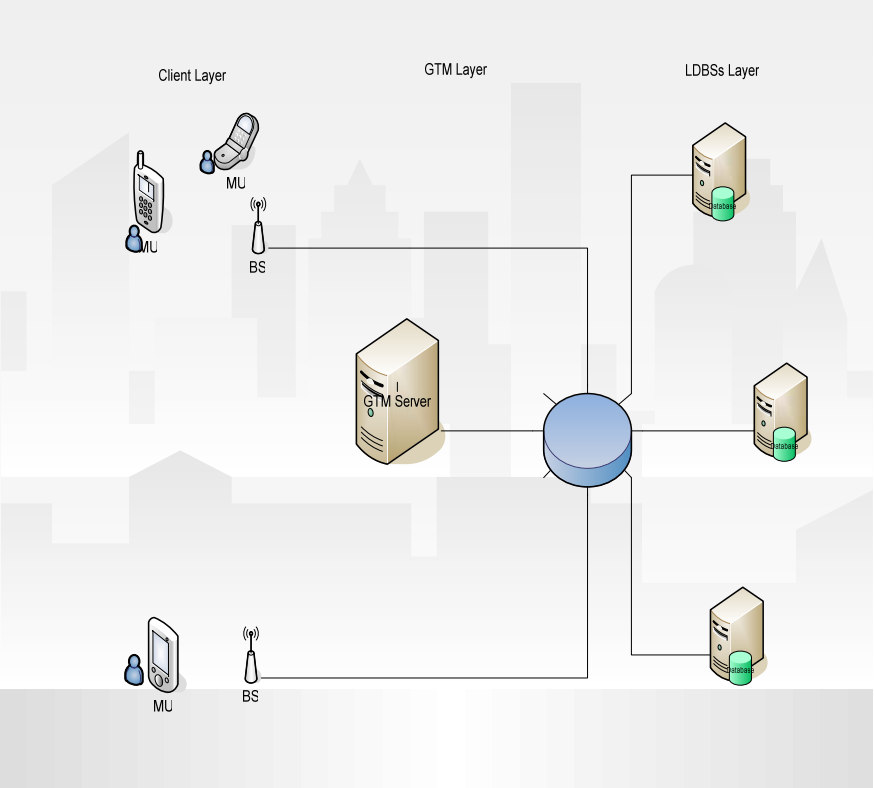


Figura 2 – Architettura hardware

Più nel dettaglio, il server si prende carico di tutta la parte computazionale, mentre il client ha l'unico compito di inviare query/update al server e prelevare i risultati da quest'ultimo a elaborazione avvenuta (query shipping).

Come già descritto in precedenza, nel sistema una Mobile Transaction coinvolge MU e FH, mentre l'esecuzione della stessa avviene totalmente su rete fissa (FH).

Esempi di applicabilità di tale scenario sono le query dipendenti dalla posizione (per esempio richiesta degli hotel presenti in un raggio di 5 KM) e le transazioni che coinvolgono un enorme set di dati.

Sfruttando questa architettura la gestione della mobilità dell'utente è un problema semplice da risolvere in quanto non è la singola BS a tenere traccia delle operazioni svolte dal singolo utente mobile, ma il tutto è gestito in maniera centralizzata.

Identificando, a livello di GTM, una transazione, all'atto del suo inizio , in maniera univoca (per esempio, attraverso la cattura dell'ID del dispositivo mobile

che l'ha generata e un identificatore proprio della transazione), il migrare da una BS (problema della mobilità) all'altra non crea problemi di gestione, in quanto tutta l'elaborazione avviene sul Server Centrale.
Il GTM è il gestore della risorsa LDBS e sfrutta le procedure di gestione costituite da una base dati d'appoggio caratterizzata dalle tabelle:

- **XT**
- **XVIR_CODA**
- **XLETTO**
- **MOBILETRANSACTION**

che permettono di memorizzare le risorse che occorrono per l'esecuzione dell'algoritmo di riconciliazione.
La base dati d'appoggio è attiva in quanto utilizza i trigger seguenti, per l'implementazione del GTM:

- **INSERT_XVIR**
- **UPDATE_XVIR**
- **COUNT**
- **COUNTD**
- **STATO**
- **SHIFT**
- **SHIFT_DELETE**

Per la gestione più efficace dei trigger stessi la base dati utilizza le procedure seguenti:

- ***ModifcaXvir_coda***
- ***Modifica***
- ***Cancella***

Il GTM ha delle procedure di accesso che permettono di gestire la base dati d'appoggio e delle procedure di gestione per la base dati attiva, ovvero dei trigger per la realizzazione della logica del GTM.
L'applicazione client sviluppata nel prototipo permette su una semplice base dati effettiva di visualizzare e modificare i dati in maniera concorrente.

Un'applicazione del prototipo è per esempio quella dell'acquisto di merci da parte di utenti mobili che accedono in concorrenza sulle medesime risorse. Inoltre alla stessa base dati vi accedono anche i gestori del magazzino per cui lo scenario permette anche di testare i concetti di compatibilità tra operazioni delle transazioni descritti nel modello teorico.
Si è pensato di far gestire la concorrenza alla base dati attiva solo per semplificare l'implementazione dello scheduler transazionale che altrimenti si sarebbe dovuto far carico della gestione multithread delle transazioni.

3 Base di dati attiva per l'implementazione del GTM

In analogia a quello che avviene nella gestione delle risorse in un sistema operativo, il nostro GTM deve essere visto come un gestore delle risorse, dove le risorse sono rappresentate dagli LDBS.

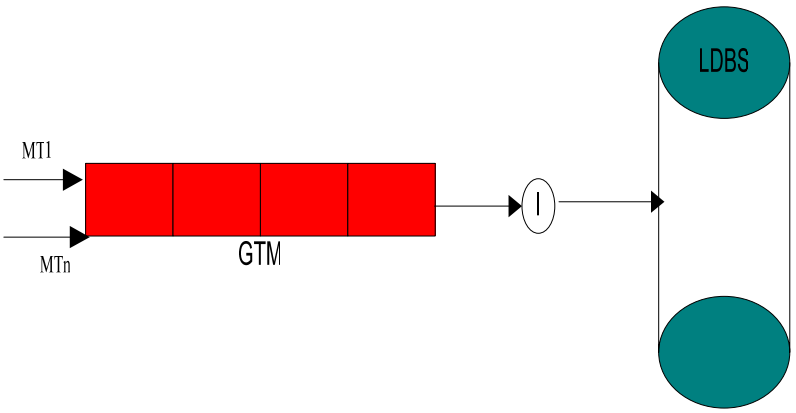


Figura 3 – Gestore delle risorse

La risorsa LDBS è globale o comune alle varie MT.
Un' applicazione qualsiasi che vuole interagire con gli LDBS non lo può fare direttamente ma deve per forza interfacciarsi con il GTM anche se la richiesta è costituita da una semplice select. Questo perché per come si è modellato il sistema la richiesta può raggiungere la base dati effettiva solo attraverso le procedure di accesso del GTM. Le procedure di gestione del GTM sono invece costituite da quelle messe a disposizione dalla base dati d'appoggio (e attiva) prevista dal nostro modello.

Queste utilizzate correttamente permettono di mantenere in uno stato coerente la base dati effettiva, ovvero i dati estratti dalla base dati attiva permettono di eseguire l'algoritmo di riconciliazione.
Nel nostro caso il GTM nel processo di schedulazione delle transazioni per renderle serializzabili rende disponibili le seguenti procedure di accesso:

- select
- selectforupdate
- update
- insert
- delete

per l'aggiornamento degli LDBS.
E' il GTM che nel nostro caso provvede a rendere visibile la base dati effettiva alle MT in modo che esse ne possano richiedere le procedure di accesso e le procedure di gestione.
Inoltre nel nostro caso l'allocazione delle risorse alla transazione è dinamica in quanto il GTM alloca e dealloca una risorsa durante l'evoluzione della MT.
Nel nostro caso il gestore è il GTM.
Entrando più nel dettaglio i compiti del gestore sono:

- consentire alle transazioni che desiderano usare risorse comuni di risolvere il problema della ***concorrenza*** per garantire il corretto utilizzo delle risorse;
- eseguire l'operazione sulla risorsa per conto delle transazioni.

Nel nostro caso il problema della concorrenza è risolto mediante il fitto utilizzo delle procedure di gestione, ovvero dei dati memorizzati nella base dati attiva per l'applicazione dell'algoritmo di riconciliazione descritto in precedenza.
Il GTM provvede a garantire la coerenza tra i dati effettivi e quelli di appoggio anche nel caso di operazioni compatibili delle transazioni eseguite in contemporanea ed offre esternamente una virtualizzazione di un LDBS.
Nel caso di operazioni incompatibili comunque si fa uso delle procedure di gestione del GTM anche se ciò equivale all'applicazione del classico protocollo di 2PL.

4 Sviluppo dei casi d'uso

In questo paragrafo vengono riportati alcuni casi d'uso per il sistema sviluppata al fine di comprendere le funzionalità offerte dai vari moduli software dal punto di vista dell'utente.

Ricordiamo brevemente che i casi d'uso sono descrizioni di come può essere utilizzato un sistema. Si tratta, essenzialmente, di una tecnica per scoprire, chiarire e concordare i requisiti di un sistema.

L'applicazione client sviluppata ovviamente costituisce solo un prototipo usato per il testing del sistema e può essere particolarizzata in base alle esigenze del client. Per cui i casi d'uso vogliono solo chiarire come funzionerà il sistema nell'ottica della gestione della concorrenza mediante l'applicazione di una variante del protocollo di 2PL.

Definire i casi d'uso significa:

- Individuare chi (persone, o altri sistemi "esterni") dovrà utilizzare il sistema
- Chiedersi quali sono gli obiettivi che intendono conseguire utilizzando il sistema (all'incirca, un caso d'uso per ogni obiettivo)
- Approfondire, in termini di descrizione di scenari concreti, ciascuna modalità di utilizzo, chiarendo il modo in cui inizia, le risposte che l'utilizzatore si attende dal sistema, la sequenza di passi con cui l'interazione si svolge, eventuali altri soggetti (esterni al sistema) coinvolti.

Ragionare con i committenti (e le altre parti interessate al sistema, come gli utilizzatori) in termini di casi d'uso (cioè di "storie" concrete di utilizzo) agevola notevolmente la scoperta dei requisiti ed il loro progressivo chiarimento. E costituisce un ottimo punto di partenza per le attività di analisi e design, implementazione e test del sistema.

4.1 Individuazione degli attori

Client: utente dotato di un dispositivo portatile con connettività wireless interessato a dati contenuti su databases remoti ai quali può accedere utilizzando come intermediario il GTM installato su Server Centrale.

4.2 Individuazione dei casi d'uso

I casi d'uso individuati sono i seguenti:

Avvia operazioni

VisualizzaDati

SelectforUpdate

ModificaDati

ConfermaModifica

Termina Operazione

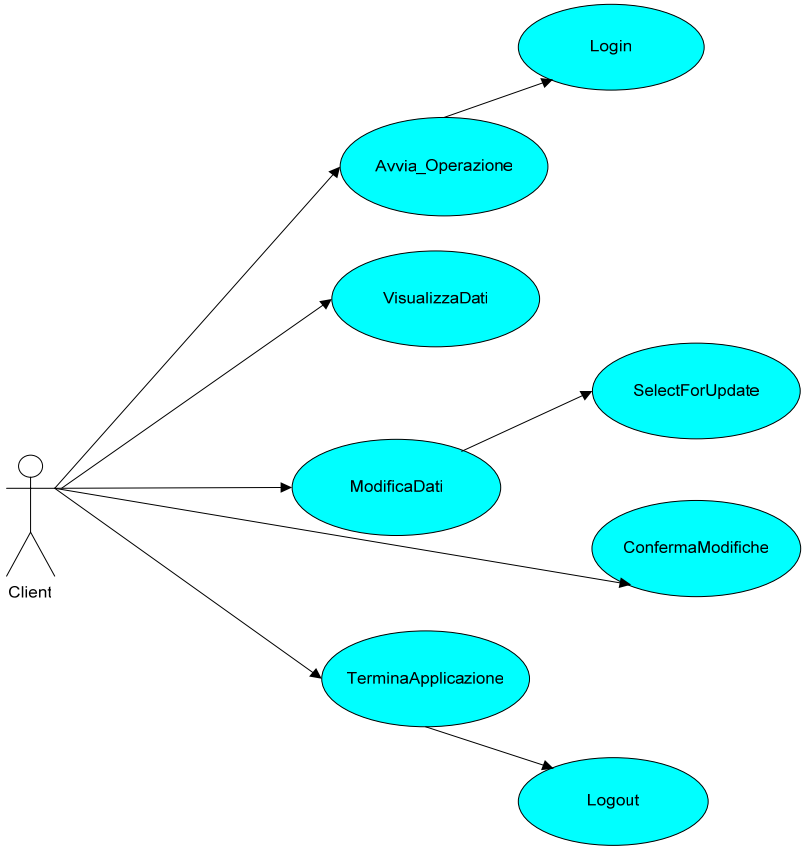


Figura 4 – Casi d'uso possibili dell'applicazione

4.2.1 Caso d'uso: Avvia operazione

Pre-condizioni: L'utente accende il proprio dispositivo portatile e manda in esecuzione l'applicazione client, il GTM è in ascolto su server e pronto a gestire una richiesta di connessione.

- l'utente avvia l'applicazione di testing e verfica del prototipo
- l'applicazione identifica il dispositivo
- l'utente fornisce i parametri di autenticazione richiesti dal software
- viene creata la comunicazione con il GTM su Server
- il GTM dopo appositi controlli consente l'accesso dell'utente

Post-condizioni:il sistema è pronto ad eseguire le richieste dell'utente

Scenari-alternativi:

- al punto 5 il GTM nega l'accesso all'utente
  viene visualizzato un messaggio di errore e l'utente è invitato a rieffettuare il Login
- disconnessione
  viene visualizzato un messaggio di errore e l'utente è invitato a rieffettuare il Login a riconnessione stabilita

4.2.2 Caso d'uso:VisualizzaDati

Pre-condizioni:Il Login dell'utente è andato a buon fine e la connessione con il GTM è attiva

- l'utente invia la query al GTM
- il GTM esegue la query su DB e invia il risultato al client
- il client riceve e visualizza il risultato della query

Post-condizioni:il sistema è pronto ad eseguire ulteriori richieste dell'utente
Scenari-alternativi:

disconnessione

- viene visualizzato un messaggio di errore e l'utente è invitato a rieffettuare l'invio della query a riconnessione stabilita

4.2.3 Caso d'uso:SelectForUpdate

Pre-condizioni:Applicazione client e GTM operanti,comunicazione attiva tra client e GTM

- l'utente invia al GTM una query ed una richiesta di lock
- il GTM effettuerà delle operazioni sulle tabelle di appoggio per gestire la richiesta di lock
- il GTM invia all'utente il risultato della query una volta che la Mobile Transaction detiene i lock richiesti, a meno che la richiesta non possa essere asservita.
- l'applicazione client riceve e visualizza il risultato della query.

Post-condizioni: il sistema è pronto ad eseguire modifiche su i dati appena forniti
Scenari-alternativi:
disconnessione

- l'utente viene avvertito della disconnessione con un messaggio di errore
- il GTM gestirà lo stato della Mobile Transaction nella fase di disconnessione (potrebbe abortirla o meno)
- l'utente, a riconnessione avvenuta, invia al GTM una richiesta di controllo
- il GTM effettuerà un controllo per verificare che la Mobile Transaction possa riprendere o meno
- il GTM invia all'utente l'esito del controllo

4.2.4 Caso d'uso: Modifica Dati

Pre-condizioni:l'utente ha effettuato la SelectForUpdate,la connessione con il GTM è attiva.

- l'utente comunica al GTM le operazioni,specificando la loro classe d'appartenenza, da effettuare su i dati restituiti dalla SelectForUpdate
- il GTM esegue le modifiche sulle copie dei dati memorizzati in tabelle di appoggio
- il GTM comunica al client l'avvenuta esecuzione delle operazioni

Post-condizioni:il sistema è pronto ad eseguire ulteriori richieste dell'utente

Scenari-alternativi:
disconnessione

- l'utente viene avvertito della disconnessione con un messaggio di errore
- il GTM gestirà lo stato della Mobile Transaction nella fase di disconnessione(potrebbe abortirla o meno)
- l'utente,a riconnessione avvenuta,invia al GTM una richiesta di controllo
- il GTM effettuerà un controllo per verificare che la MobileTransaction possa riprendere o meno
- il GTM invia all'utente l'esito del controllo

4.2.5 Caso d'uso: Conferma Operazioni

Pre-condizioni: l'applicazione client e il GTM sono attivi, l'utente ha eseguito varie operazioni all'interno della Mobile Transaction, la comunicazione tra GTM e Client è attiva.

- l'utente invia il comando di COMMIT al GTM
- il GTM sfruttando le tabelle di appoggio ed appositi algoritmi di riconciliazione, in modo tale che l'esecuzione concorrente di una

collezione di transazioni produca un risultato che si potrebbe ottenere con una esecuzione sequenziale, genera la transazione che deve essere eseguita su DB per aggiornarne i dati

- Il GTM comunica al client l'esito della transazione

Post-condizioni:l'applicazione client offre all'utente la possibilità di cominciare un'altra transazione o terminare l'applicazione.

4.2.6 Caso d'uso: Termina Applicazione

Pre-condizioni: L'applicazione client è attiva

- l'utente sceglie di effettuare il log-out e invia la richiesta al GTM
- Il GTM controlla che non vi siano MobileTransaction attive relative all'utente
- Nel caso la Mobile Transaction relativa all'utente fosse ancora attiva la manda in abort e aggiorna le tabelle di appoggio in maniera consona

Post-condizioni:l'applicazione client viene chiusa

5 Requisiti funzionali del GTM

Il modulo in questione deve offrire un set di funzionalità base che consenta da un lato la gestione della concorrenza delle transazioni, e dall'altro l'amministrazione dell'intero sistema.

Insieme a tale set di funzionalità base, è richiesta poi l'integrazione di un insieme di servizi di supporto per la consistenza dei dati del database. Particolare importanza assumono in tal senso i tool classici per garantire le proprietà di atomicità, consistenza, isolamento e durability.

In particolare tale modulo, da un punto di vista funzionale ,dovrà offrire una serie di primitive per la gestione dell'accesso a dati condivisi da parte di più transazioni contemporaneamente attive.

Il GTM (Global Transaction Manager) è il cuore del sistema perché permette di:

- gestire il login dell'utente mobile;
- gestire la concorrenza delle transazioni basandosi sulla semantica delle stesse;
- gestire le condizioni di deadlock;
- gestire il problema della starvation legato alle classi di compatibilità definite per le operazioni delle transazioni;
- gestire le operazioni di select sottoposte dal client;
- gestire le query di selectforupdate (ovvero query di selezione che, sottoposte dal client, sono finalizzate all'aggiornamento dei dati letti);
- gestire la momentanea disconnessione del client;
- gestire le query di update sottoposte dal client;

Quindi il GTM permette di comunicare all'applicazione client il buon esito delle transazioni oltre che l'invio dei "resultsets" delle query di aggiornamento.
E' il GTM che si interfaccia con i vari LDBS indipendentemente dalla loro dislocazione fisica e permette di interagire con la base dati d'appoggio. Quest'ultima permette di gestire le risorse virtuali che nascono con lo scopo di aumentare la concorrenza.
Il GTM permette ai vari dispositivi mobili di operare in modo concorrente nel caso di operazioni delle transazioni compatibili ed è necessariamente "multi-thread" visto che deve poter servire in contemporanea più client.
L'applicazione client deve conoscere l'indirizzo del server ed in particolare il protocollo di comunicazione del server che nel nostro caso è il TCP/IP Transmission Control Protocol e l'IP (Internet Protocol).
Un client quindi per comunicare con un server dovrà per prima cosa creare una socket con tale server, specificando l'indirizzo IP della macchina su cui il server è in ascolto.

Il GTM, progettato secondo la classica metodologia OO (Object Oriented) è diviso in 5 parti:

- **Comunication manager**: si occupa della comunicazione client-server, stream, lettura ingressi e invio uscite per l'applicazione client;
- **State_Transaction_Oracle**: supervisiona lo stato delle transazioni**;**
- **Transaction Manager**: è lo scheduler e gestore delle transazioni - permette di effettuare il login dell'applicazione client, di restituire il risultato della select, di restituire il risultato della selectforupdate, di recuperare dalle tabelle di appoggio tutti i dati necessari per poter apportare l'update definitivo sul DB e di effettuare gli update sui campi bloccati in precedenza dalla selectforupdate, garantendo la serializzabilità per gli accessi concorrenti sulle stesse risorse;
- **DBMS Connection:** è la classe che permette la gestione della connessione al database effettivo e a quello di appoggio;
- **Application Manager**: rappresenta il gestore dell'applicazione ovvero si occupa di richiamare i diversi metodi in base al comando ricevuto dall'applicazione client;

Nelle figure sono riportati dei sequence diagram che descrivono, nello scenario ipotizzato, il login di un nuovo utente con il suo palmare, la richiesta di selectforupdate dell'utente e la richiesta di chiusura della transazione dell'utente.

Si è ipotizzata l'esistenza:

- di un oggetto di tipo "ClientManager", capace di gestire le richieste concorrenti degli utenti mobili;
- di un oggetto di tipo "ComunicationManager" capace di gestire la comunicazione tra applicazione client e GTM;
- di un oggetto di tipo "ApplicationManager" per la gestione delle operazioni tipiche della particolare applicazione;
- di un oggetto di tipo "TransactionManager" per la gestione delle transazioni;
- di un oggetto di tipo "DBInterface" per l'interfacciamento col database.

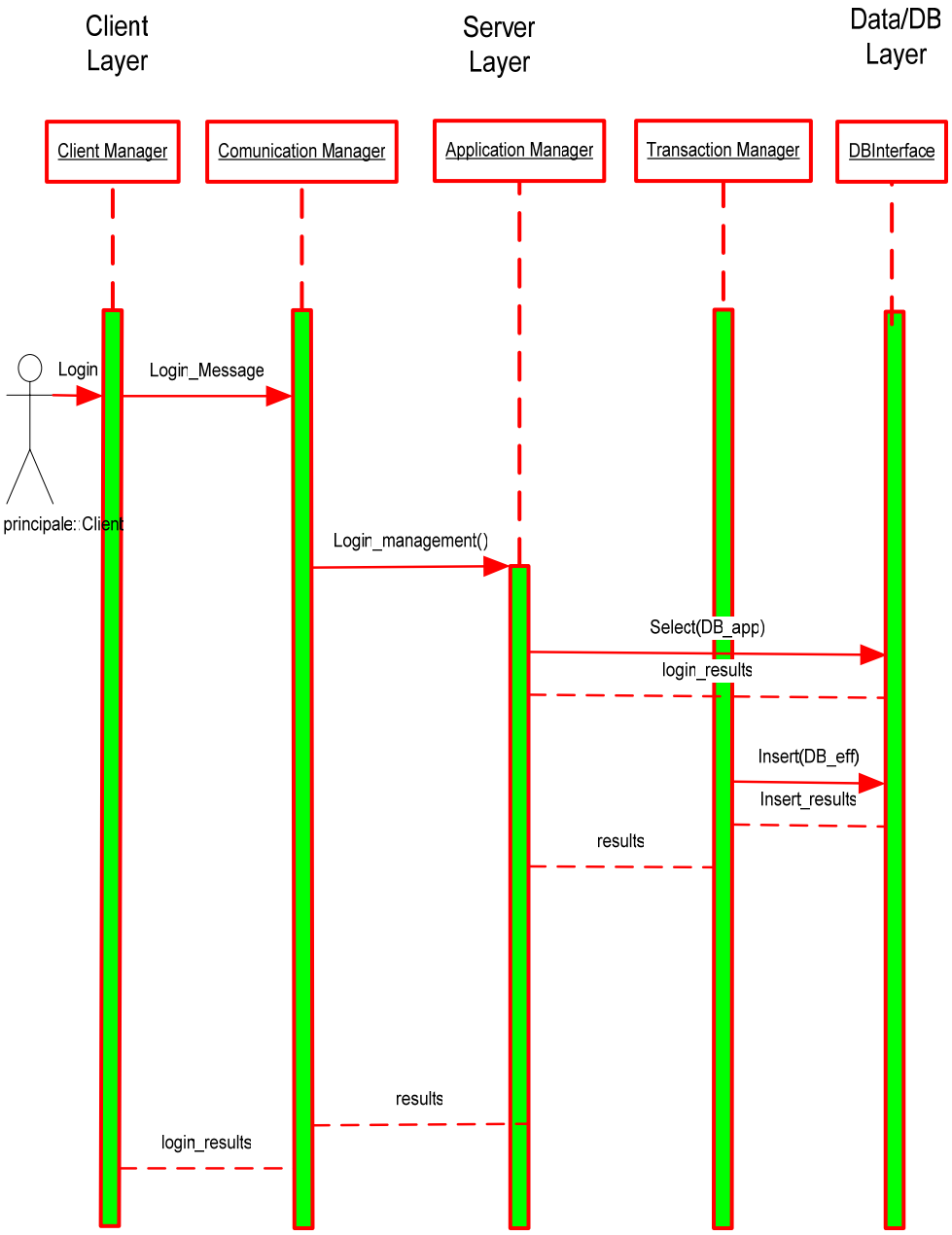


Figura 5 – Sequence diagram relativo al Login di un nuovo utente mobile

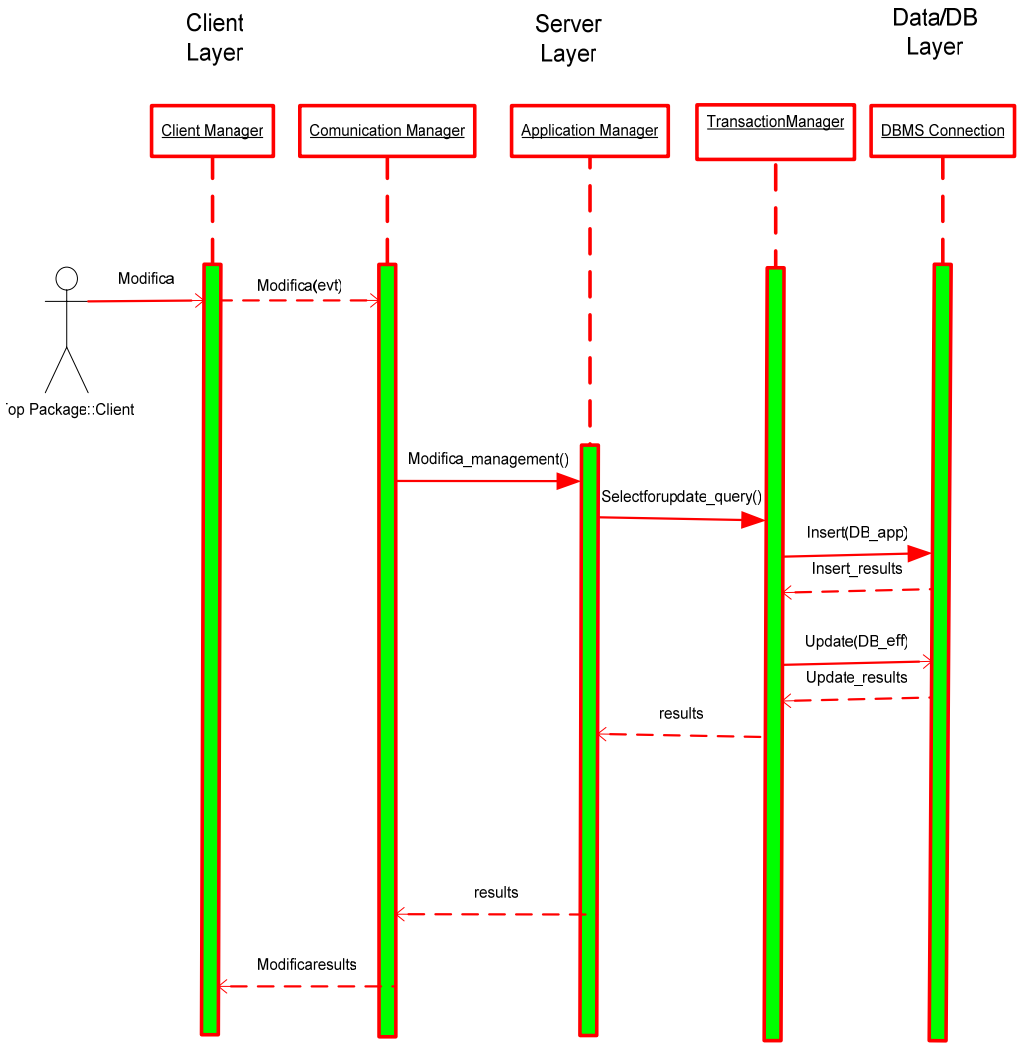


Figura 6 – Sequence diagram relativo al Modifica di un utente mobile

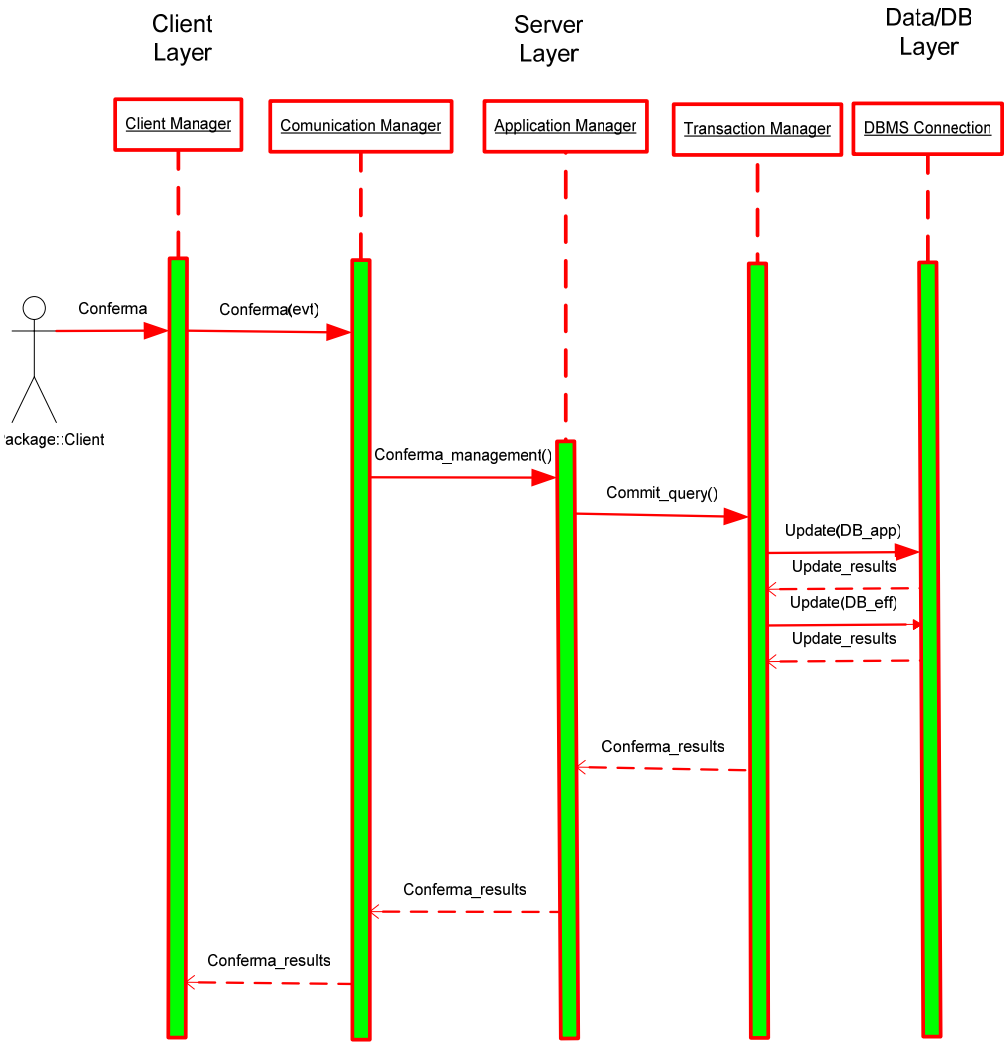


Figura 7 – Sequenze diagram relativo al Commit sottoposto dall'utente mobile

6 Architettura software di base

Alla luce di queste considerazioni l'architettura di massima del sistema può essere espansa attraverso il component diagram mostrato in figura:

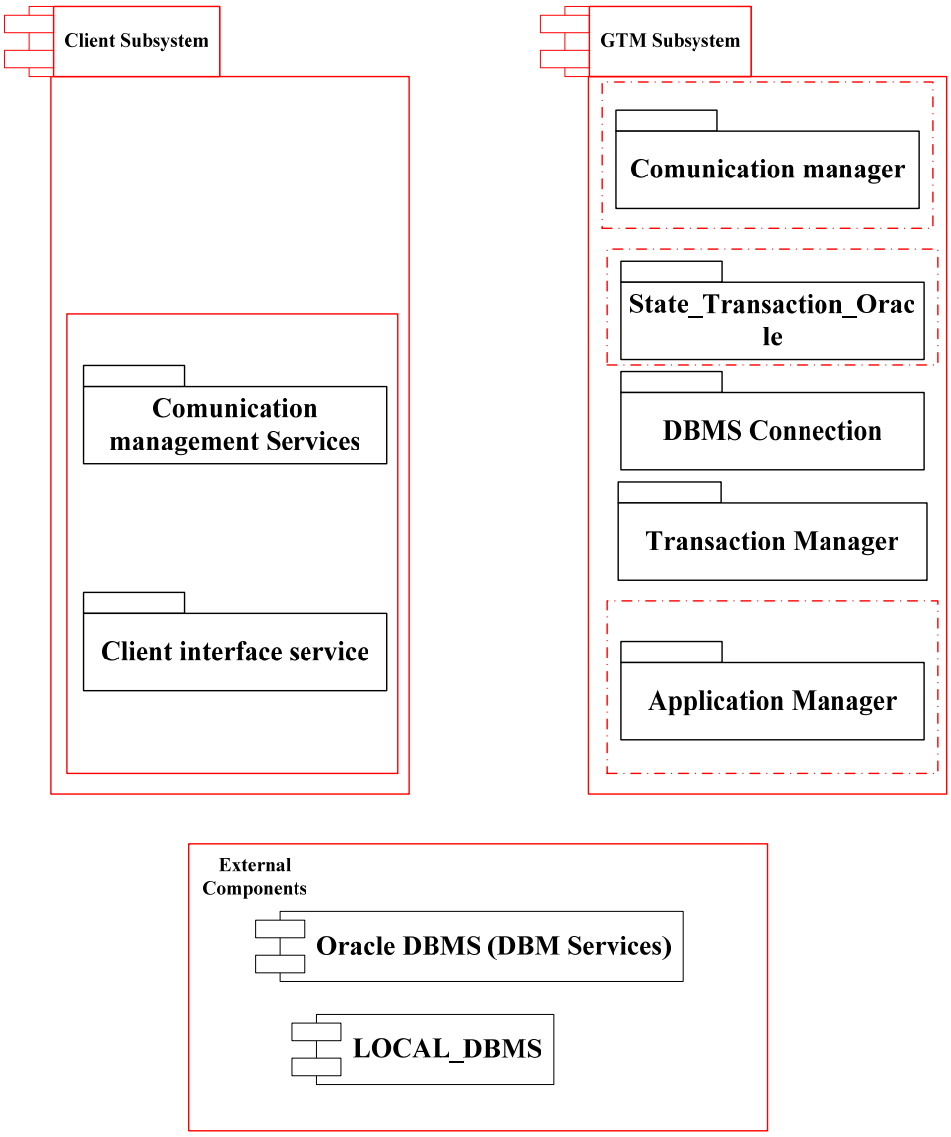


Figura 8 – Component Diagram del sistema

L'architettura del sistema sarà quindi di tipo "multi-thread", di cui evidenziamo i 3 livelli funzionali:

- *livello Client:* con interfaccia grafica per palmare;
- *livello GTM :* middleware;
- *livello external components:* base dati effettiva e base dati d'appoggio;

In particolare come si può notare della figura, il livello GTM dovrà essere caratterizzato da una serie di funzionalità richiamabili dal Front-end che garantiscano tutte le funzionalità di un GTM avanzato. A questo livello può ritroviamo un insieme di servizi dedicati e di servizi di base, tra cui evidenziamo:

- ***Transaction Manager***: permette l'esecuzione di transazioni gestendo la loro concorrenza sulla base della compatibilità delle loro operazioni.
- ***Comunication manager***: questi servizi permettono la comunicazione dei vari client col server multithread grazie al protocollo TCP/IP.
- ***Application Manager***: questi servizi sono caratteristici della singola applicazione e ne permettono l'esecuzione.
- ***Client Interface Services***: sono i servizi tipici di interfaccia delle tipiche applicazioni client.
- ***DBMS Connection***: rappresentano i servizi di connessione a database sia esso di appoggio per la gestione delle repliche degli oggetti coinvolti nelle Mobile transactions che effettivo. Il collegamento a database avviene mediante i driver nativi JDBC per l'accesso ad un qualsiasi database.

7 Rappresentazione delle strutture dati del gestore GTM

Per semplificare l'implementazione del GTM è stata utilizzata una base dati attiva basata sul concetto di trigger. Attraverso tale metodologia è stato sia garantito l'accesso serializzato alle strutture dati condivise dello schedulatore (gestore di risorse), ma anche la possibilità di semplificarne notevolmente la complessità, sfruttando il paradigma evento-azione-reazione dei trigger.
Il GTM alla nascita di una Mobile transaction inserisce una tupla nella tabella d'appoggio MobileTransaction che contiene le informazioni relativa alla mobile transaction generata.

MOBILETRANSACTION

La tabella MobileTransaction è costituita dai campi seguenti:

ID univoco della Mobile Transaction;
ID_DISPOSITIVO che ha generato la Mobile Transaction;
STATO della Mobile Transaction che appena nata sarà Run;
COUNT che tiene conto del numero di thread creati per la generica risorsa e risulta maggiore di uno solo se il client si ricollega in seguito ad una disconnessione, per cui permette l'aggiornamento corretto della tabella MOBILLETRANSACTION.

Nome	Nullo?	Tipo
ID	NOT NULL	NUMBER
ID_DISPOSITIVO		NUMBER
STATO		NUMBER
COUNT		NUMBER

Figura 9 Tabella MOBILLETRANSACTION della base dati attiva

Il GTM poi nella tabella Xvir memorizza le informazioni relative alle risorse su cui operano le varie Mobile Transaction.

XVIR

Per ogni risorsa il GTM memorizza la Tabella e la Tupla a cui appartiene (nei campi TAB e ID_TUP), il Campo e il Valore su cui stiamo operando (nei campi CAMPO e VALORE), il numero di transazioni che stanno agendo simultaneamente sulla risorsa (COUNT = numero di transazioni attive COUNTD = numero di transazioni che non hanno ancora eseguito il commit ma il dispositivo che le ha generate si è disconnesso), lo STATO della risorsa che può essere BUSY (0) se vi è almeno una transazione attiva su di essa, o ,BUSY_D (1) se le transazioni che operano su di essa sono disconnesse.
Se la risorsa non è presente in XVIR vuol dire che è totalmente FREE.
In CLASSE è invece memorizzata la classe di compatibilità delle operazioni delle transazioni che stanno agendo sulla risorsa.
In STARVATION vi è un valore booleano che viene opportunamente aggiornato per risolvere il problema che transazioni con operazioni incompatibili vengano penalizzate eccessivamente a vantaggio di più transazioni che vogliono effettuare operazioni compatibili.

Nome	Nullo?	Tipo
ID	NOT NULL	NUMBER
MT		NUMBER
TAB		VARCHAR2
ID_TUP		NUMBER
CAMPO		VARCHAR2
VALORE		NUMBER
CLASSE		NUMBER
COUNT		NUMBER
COUNTD		NUMBER
STATO		NUMBER
STARVATION		NUMBER

Figura 10 Tabella XVIR della base dati attiva

Altre strutture dati dello schedulatore sono Xletto, XT.

XLETTO, XT

In XLETTO il GTM memorizza i valori delle risorse all'atto dell'acquisizione di quest'ultime da parte delle Mobile Transactions.
In XT invece i valori delle risorse seguono l'evoluzione così come dettata dall'esecuzione delle Mobile Transaction prima del commit.

Nome	Nullo?	Tipo
IDXVIR	NOT NULL	NUMBER
MT	NOT NULL	NUMBER
VALORE		NUMBER

Figura 11 Tabelle XLETTO e XT della base dati attiva

In XVIR_CODA il GTM gestisce la coda delle Mobile Transaction sulle varie risorse e quindi può bloccare le operazioni non compatibili e far procedere in concorrenza le operazioni compatibili. Il campo POSIZIONE, in abbinamento al campo CLASSE, permette di capire che priorità di servizio ha la generica Mobile transaction. Lo STATO mi permette di capire in che condizioni sta operando la generica Mobile transaction.

XVIR_CODA

Nome	Nullo?	Tipo
IDXVIR	NOT NULL	NUMBER
MT	NOT NULL	NUMBER
POSIZIONE		NUMBER
CLASSE		NUMBER
STATO		NUMBER

Figura 12 Tabella XVIR_CODA della base dati attiva

8 Procedure di gestione della base dati attiva

Abbiamo adottato la seguente codifica degli stati di una mobile transaction:

Codifica	*Stato*
0	**RUN**
1	**WAIT**
2	**DISCONNECTED**
3	**ABORT**
4	**COMMIT**

Figura 13 Codifica degli stati delle mobile transaction

Di seguito viene chiarito il funzionamento dei trigger utilizzati per la gestione della base dati attiva.

TRIGGER INSERT_XVIR (AFTER INSERT ON XVIR)

Questo trigger permette di inserire nelle tabelle XLETTO, XT, XVIR_CODA le tuple in seguito all'inserimento delle tuple in XVIR.

In particolare, in seguito ad un inserimento su XVIR il TRIGGER provvede ad eseguire INSERT opportuni nelle tabelle XLETTO,XT,XVIR_CODA. Ovviamente l'inserimento in queste è relativo agli ultimi valori inseriti in XVIR stessa e per questo si utilizzano delle variabili di appoggio idxvir, st ,valore e classe nelle quali con l'ausilio del costrutto PLSQL **new.nome_campo** si inseriscono i valori aggiornati dei campi in questione delle tabelle di appoggio in cui inserire le nuove tuple, ovvero l'identificativo dell'ultima tupla inserita in xvir_coda, l'identificativo della transazione, il valore della risorsa richiesta e la classe di appartenenza della transazione.

TRIGGER UPDATE_XVIR
AFTER UPDATE OF MT ON XVIR

Questo trigger inserisce nelle tabelle XLETTO, XT, XVIR_CODA le tuple relative in seguito ad una modifica di MT della tabella XVIR.

In particolare quando una Mobile Transaction vuole operare su una risorsa già occupata da una transazione con essa compatibile il campo MT di XVIR verrà aggiornato con l'identificativo dell'ultima Mobile Transaction giunta. Anche questo aggiornamento dovrà far scattare un trigger che opererà allo stesso modo del TRIGGER INSERT_XVIR. Anche in questo caso si utilizzano delle variabili di appoggio *idxvir*, *st*, *valore* e *classe* nelle quali con l'ausilio del costrutto PLSQL **new.nome_campo** si inseriscono i valori aggiornati dei campi, ovvero l'identificativo dell'ultima tupla modificata in xvir_coda, l'identificativo della transazione, il valore della risorsa richiesta e la classe di appartenenza della transazione.

TRIGGER COUNT
AFTER UPDATE OF count,countd ON xvir

Questo trigger mantiene coerenti le tuple delle varie tabelle in seguito all'aggiornamento del numero di transazioni concorrenti attive o disconnesse su una risorsa.

In particolare il trigger COUNT viene richiamato ogni qualvolta una transazione utilizza la risorsa (*count*++), effettua il commit o l'abort (*count*--), si disconnette (*count--;countd*++) oppure dopo una disconnessione si ricollega (*countd--;count++)*
In seguito ad un aggiornamento del campo COUNT o COUNTD della tabella XVIR partirà il trigger che controllerà:
1)Se COUNT=0 e COUNTD=0 verrà eliminata la tupla corrispondente perchè vorrà dire che nessun altra transazione sta operando su questa risorsa
2)Se COUNT=0 e COUNTD>0 verrà invece modificato lo stato della risorsa e messo a FREE (1), ovvero sulla risorsa stanno operando solo transazioni disconnesse.
Questo trigger funziona grazie all'ausilio di cursori per la gestione di oggetti del tipo riga di Xvir che vengono selezionati nei due casi distinti evidenziati e di una procedura esterna di cancella che permette di eliminare le tuple nel primo caso.
Il cursore *CursorTupleCanc* consente di individuare eventuali tuple da cancellare nella tabella xvir per fare ciò verifica che non ci siano transazioni che stanno lavorando sulla risorsa (*count*=0) o che si siano temporaneamente disconnesse (countd=0).
Il cursore *CursorTupleFree* consente di individuare eventuali risorse da considerare free (stato=1) per fare ciò verifica che non ci siano transazioni che stanno lavorando sulla risorsa (*count*=0), ma solo transazioni temporaneamente disconnesse (*countd*>0).
Per tutte le tuple individuate dal cursore *CursorTupleCanc* richiamo la procedura cancella che elimina le tuple in questione. La tupla è identificata dalla variabile id.

Per tutte le tuple individuate dal cursore *CursorTupleFree* richiamo la procedura modifica che cambia l'attributo stato della risorsa da lock (stato=0) a free (stato=1).La tupla da modificare è identificata dalla variabile id.

TRIGGER COUNTD
AFTER UPDATE OF stato ON xvir_coda

Questo trigger aggiorna i campi relativi al numero di transazioni attive e disconnesse, della tabella xvir, in base alle modifiche effettuate sullo stato delle transazioni nella tabella xvir_coda.

In particolare, se una Mobile Transaction passa nello stato disconnesso bisogna decrementare il campo COUNT relativo alle risorse che trattiene ed incrementare il campo COUNTD.
Se una Mobile Transaction dallo stato Disconnesso passa nello stato di Abort bisogna decrementare il campo Countd relativo alle risorse su cui opera.
Se una Mobile Transaction dallo stato Disconnesso passa nello stato di Run bisogna incrementare il campo COUNT e decrementare COUNTD relativo alle risorse su cui opera.
In analogia al caso dei trigger INSERT_XVIR ed UPDATE_XVIR viene utilizzata l'istruzione PLSQL che permette di porre nelle condizioni degli elseif i valori degli stati precedenti dei vari campi delle tabelle di appoggio in questione. Questa istruzione è **old.nome_campo.** I controlli che effettua il trigger sono i seguenti:

- Se lo stato della transazione in xvir_coda è stato settato ad abort viene modificato il campo count nella tabella Mobiletransaction;
- Se la transazione è stata settata a disconnessa e si trova nella prima posizione della coda aggiorna, in xvir, il numero di transazioni attive e disconnesse (*count*-- e *countd*++);
- Se la transazione dallo stato disconnesso è passata alla stato abort ed è passata dalla posizione 1 alla posizione 0 aggiorna il numero di transazioni disconnesse nella tabella xvir (*countd*--);

- Se la transazione dallo stato disconnesso è passata alla stato run ed è nella posizione 1 aggiorna, in xvir, il numero di transazioni attive e disconnesse (*count*++ e *countd*--);
- Se la transazione dallo stato run è passata alla stato abort ed era nella posizione 1 aggiorna, in xvir, il numero di transazioni attive (*count*--) ;

TRIGGER STATO
AFTER UPDATE OF STATO ON MOBILETRANSACION

Il trigger stato mantiene la corrispondenza dello stato memorizzato nelle tabelle Mobiletransaction e xvir_coda.

In particolare il trigger stato per tutte le tuple di MobileTransaction che si modificano grazie a un cursore (*CursorStato* contenente tutte le tuple di Mobiletransaction sulle quali si è effettuata la modifica dello stato). Per ogni tupla del cursore si richiama la procedura *modificaxvir_coda* (passandole il nuovo stato) e quindi per ogni transazione selezionata attraverso il cursore si memorizza lo stato presente in MobileTransaction nella tabella xvir_coda.

TRIGGER SHIFT
AFTER UPDATE ON XVIR_CODA

Questo trigger permette di mantenere coerente lo stato delle transazioni in coda a seguito di disconnessioni delle stesse.

Questo trigger serve per:

- determinare le transazioni che si sono disconnesse e sono nella posizione 1;
- verificare se non ci sono altre transazioni in wait o in run in posizione 1, ed effettua lo shift delle transazioni in coda sulle stesse risorse;

Quindi in sostanza si attua l'aggiornamento del campo posizione in Xvir_coda. E' utilizzato un cursore che contiene tuple il cui stato è settato a disconnesso (stato=2) e si trovano nella posizione 1 e per tutte le tuple presenti nel cursore verifico se le posizioni delle transazioni in coda possono scalare.
Si contano le tuple che agiscono sulla stessa risorsa delle transazioni disconnesse che si trovano in posizione 1 e sono nello stato di run o di wait si pongono nella variabile di appoggio *cont_RunWaitPos1*.
Si contano le tuple che agiscono sulla stessa risorsa delle transazione disconnessa che si trovano in posizione maggiore di 1 e sono nello stato di run o di wait e si pongono nella variabile di appoggio *cont_RunWaitPosM1*.
Se non ci sono transazioni in run o in wait nella posizione 1, ma ci sono altre transazioni che attendono in posizione >1:
si effettua un ciclo, che terminerà quando o nella prima posizione avrò delle transazioni nello stato run o wait, oppure quando si avranno delle transazioni nello stato di disconnesso senza altre transazioni in coda per quella stessa risorsa, in cui:

- si settano ad abort eventuali transazioni che si erano disconnesse nella posizione 1;
- si scalano tutte le posizioni delle transazioni in coda sulla risorsa considerata;
- si ricalcola il numero di transazioni in run o wait nella posizione 1 e il numero di transazioni che attendono in coda.

Si utilizza una variabile di appoggio del tipo riga di Xvir_coda .

TRIGGER SHIFT_DELETE
AFTER UPDATE ON XVIR_CODA

Questo trigger permette di mantenere coerente lo stato delle transazioni in coda a seguito dell'esecuzione del commit o dell'abort delle stesse.

Questo trigger determina le transazioni che sono andate in abort o in commit ed effettua lo shift delle transazioni in coda sulle stesse risorse.

Serve per l'aggiornamento del campo posizione in Xvir_coda nel caso in cui ci sono delle transazioni nello stato di abort o di commit in posizione qualsiasi nella coda come per il precedente però le tuple in questo caso vengono cancellate da Xvir_coda invece che aggiornate nello stato di abort. Il modo di procedere in analogia al trigger precedente è basato su un doppio ciclo iterativo ed due cursori che memorizzano tutte le tuple il cui stato è stato settato ad abort (stato=3) o commit (stato=4).

- Si cancella la transazione che è andata in abort o commit dalla tabella xvir_coda;
- Se la posizione della tupla cancellata è 1 si controlla se occorre effettuare uno shift;
- Si contano le tuple che agiscono sulla stessa risorsa delle transazione cancellata che si trovano in posizione 1 e sono nello stato di run o di wait;
- Si contano le tuple che agiscono sulla stessa risorsa delle transazione cancellata che si trovano in posizione maggiore di 1 e sono nello stato di run o di wait;
- Se non ci sono transazioni in run o in wait nella posizione 1, ma ci sono altre transazioni che attendono in posizione >1, si effettua un ciclo che terminerà quando o nella prima posizione avrò delle transazioni nello stato run o wait, oppure quando si avranno delle transazioni nello stato di disconnesso senza altre transazioni in coda per quella stessa risorsa, in cui:

    si settano ad abort eventuali transazioni che si erano disconnesse nella posizione 1;

    si scalano tutte le posizioni delle transazioni in coda sulla risorsa considerata;

    si ricalcola il numero di transazioni in run o wait nella posizione 1 e il numero di transazioni che attendono in coda;

Se la posizione della tupla cancellata è maggiore di uno si scalano di posizione le eventuali transazioni in coda dietro di lei.

PROCEDURE

Procedura Cancella

Permette di cancellare da XVIR una tupla dato l'id della Mobile transaction in questione.

Procedura ModifcaXvir_coda

Permette di aggiornare lo stato nella tabella XVIR_CODA dato il nuovo stato da settare**.**

Procedura Modifca

Permette di porre a WAIT lo stato di una Mobile transaction dato il suo id.

CAPITOLO 5

Validazione sperimentale del modello e conclusioni e sviluppi futuri

In questo capitolo viene confrontato il protocollo ibrido utilizzato per la schedulazione delle transazioni rispetto al classico 2PL. A tale proposito viene condotta un'analisi atta ad esplicitare le variazioni dei tempi di esecuzione in funzione del numero di conflitti e del numero di incompatibilità delle operazioni, nonchè la percentuale di abort delle transazioni disconnesse in funzione del numero di transazioni disconnesse stesso e del numero di incompatibilità tra operazioni delle transazioni. Ovviamente in questo secondo caso per il confronto con il classico 2PL va ipotizzata una politica per la gestione delle transazioni disconnesse. In generale con il classico 2PL si forza un abort a disconnessione avvenuta in funzione di un certo timeout e si ipotizza di essere in questo caso.

1 Analisi dei tempi di esecuzione

Detto *c* il numero di conflitti, il tempo medio di esecuzione, per transazioni che adottano il protocollo di 2PL, sarà dato dalla seguente formula:

$$t_{e\,\mathrm{sec}\,2}(c) = \frac{((n-c) * t_{e\,\mathrm{sec}} + c * (t_{e\,\mathrm{sec}} + \frac{t_{e\,\mathrm{sec}}}{2}))}{n}$$

con *n* numero totale di transazioni , *c* numero di conflitti, e t_{esec} il tempo medio di esecuzione nel caso di assenza di conflitti.

Questo è frutto di ipotesi semplificative sull'istante di sovrapposizione del lavoro di due transazioni concorrenti: ovvero si ipotizza che l'arrivo della transazione concorrente è mediamente situato a metà dell'intervallo di esecuzione della precedente. Inoltre si ipotizza che non possano verificarsi conflitti multipli. Dalla precedente formula è possibile osservare che nel caso di un carico transazionale con sole transazioni non in conflitto (c=0) si ha un miglioramento del 50% del

tempo medio di esecuzione rispetto al worst case di un carico transazionale con tutte operazioni in conflitto sulla medesima risorsa ($c=n$).

Nella valutazione dei tempi medi di esecuzione per il protocollo proposto bisogna tener conto non solo del numero di transazioni in conflitto ma anche dell'incompatibilità tra le operazioni eseguite da esse.

Detti:

n = numero transazioni totali

c = numero di conflitti

i = numero di incompatibilità

la probabilità di trovare *k* conflitti incompatibili è la seguente:

$$P(k) = \frac{(C_{i,k} * C_{n-i,c-k})}{C_{n,c}}$$

con $C_{n,m} = \binom{n}{m}$

Il tempo medio di esecuzione sarà in queste ipotesi pari alla somma dei prodotti seguenti:

$$t_{me\,\sec}(c,i) = \sum_{k=o}^{\min(i,c)} P(k) * t_{e\,\sec 2}(k)$$

Valutati i tempi come indicato è possibile confrontarli con quelli del 2PL classico che sono invarianti col numero di operazioni incompatibili e crescono linearmente all'aumentare del numero di conflitti. Risulta possibile quindi valutare quando il nostro metodo migliora rispetto al 2PL classico.

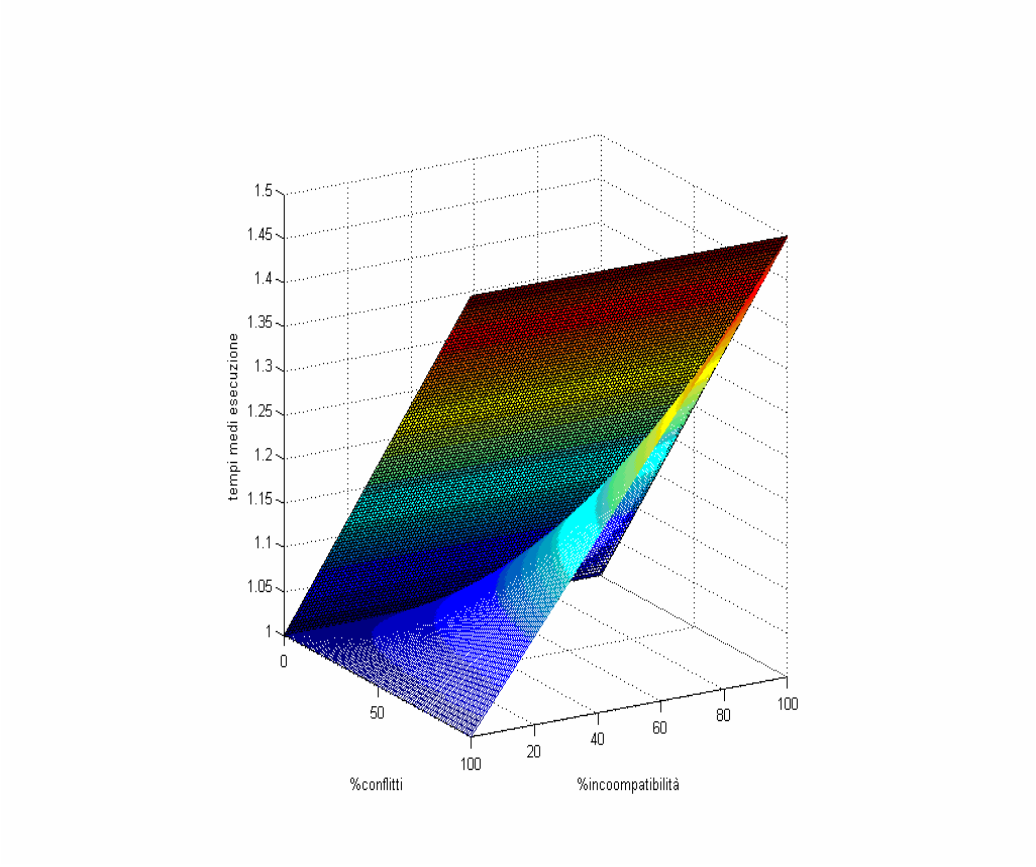


Figura 1 – Tempi medi di esecuzione in funzione di %incompatibilità e %conflitti

Dal grafico precedente si osserva, all'aumentare del numero di conflitti e della percentuale di incompatibilità, che i tempi di esecuzione del nostro metodo crescono e si mantengano inferiori a quelli relativi al 2PL classico perché nel modello non si è valutato l'overhead dell'applicativo implementato, ma si è fatta un'analisi teorica di risposta del sistema. In particolare si osserva che l'applicazione dell'approccio proposto è particolarmente conveniente nel caso di percentuali medio alte di conflitti e percentuali basse di incompatibilità, perché è in questa zona che la riduzione del tempo di esecuzione è più accennata.

2 Analisi della percentuale di abort delle transazioni disconnesse

Per l'analisi delle disconnessioni valutiamo la percentuale di abort nel caso del nostro approccio rispetto a quella del 2PL in funzione della percentuale di transazioni disconnesse e della percentuale dei conflitti delle transazioni. La percentuale di abort nel caso del 2PL standard, visto che si prescinde dalla percentuale di operazioni incompatibili, è funzione solo del timeout. L'approccio seguito ci permette di valutare la percentuale di abort delle transazioni disconnesse come il prodotto delle probabilità di disconnessione, della probabilità che siano incompatibili e per la probabilità di conflitto, essendo i tre eventi statisticamente indipendenti:

$$P(Abort) = P(disconnessione) * P(incompatibilità) * P(conflitto)$$

Di seguito è riportata la percentuale di abort in funzione delle percentuali di conflitto e di disconnessione, considerando più grafici al variare della percentuale di incompatibilità.

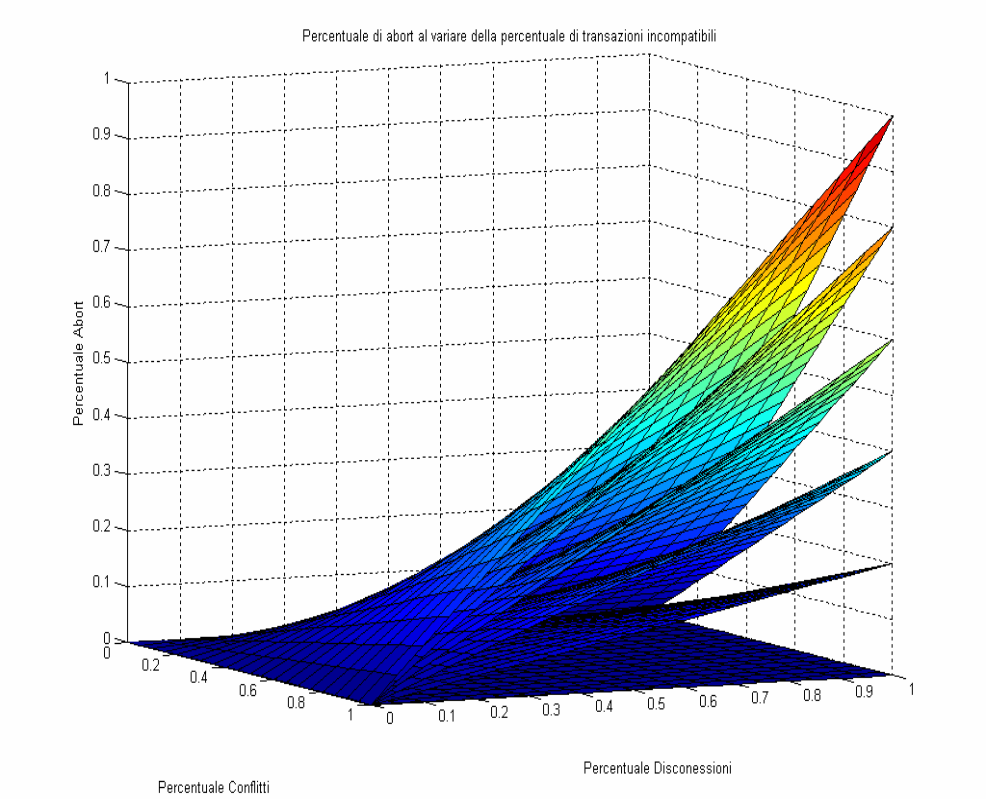


Figura 2– Tasso di abort in funzione di %conflitti e %disconnessioni

Si osserva che all'aumentare della percentuale di disconnessione nonché di quella dei conflitti, le percentuali di abort crescono soprattutto per valori delle percentuali dal 70% in poi. Inoltre la crescita in funzione delle percentuali di disconnessione e dei conflitti ed è estremamente regolare. Ciò concorda col fatto che i due eventi hanno lo stesso peso. Inoltre all'aumentare della percentuale di incompatibilità (fissata sulla generica superficie) le curve mostrano un aumento più sostenuto del tasso di abort a parità di percentuali di disconnessione e di conflitti .

3 Risultati, attesi e ottenuti, nei vari casi d'uso di un prototipo realizzato per il testing del modello

Consideriamo ora cinque casi d'uso dell'applicazione pensata per una validazione sperimentale di tipo funzionale del modello teorico proposto. L'applicazione sviluppata prevede l'acquisto di oggetti di un tipico magazzino di prodotti di elettronica (selezionando la quantità e il tipo di oggetto) e la gestione del magazzino da parte dell'amministratore che può sia aggiornare il prezzo degli articoli , sia cambiare la quantità delle scorte di magazzino.

Queste due operazioni distinte sulla base dati sono incompatibili e i vari casi d'uso metteranno in evidenza le novità dell'approccio utilizzato per la gestione della concorrenza delle operazioni. Il form utilizzato per l'identificazione del cliente è il seguente:

Figura 3 – Form dell'applicazione acquisti di prodotti elettronici (login)

Esso permette l'identificazione del cliente generico mediante username e password. L'utente dopo l'identificazione può passare alla fase di scelta del settore nel quale acquistare gli articoli (vedi figura 5):

Figura 4 – Form dell'applicazione acquisti di prodotti elettronici (scelta del settore)

E' possibile poi scegliere il settore e inoltre il bottone visualizza permette di avere la lista degli oggetti acquistabili con le relative quantità e caratteristiche degli articoli.

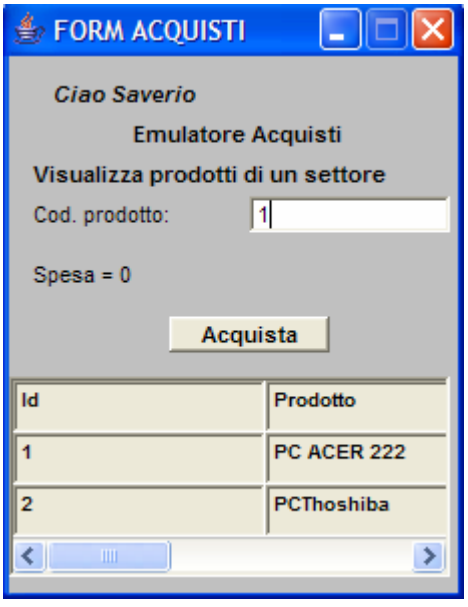


Figura 5 – Form dell'applicazione acquisti di prodotti elettronici (scelta del codice prodotto)

Nel primo caso d'uso si ipotizza la presenza di quattro utenti distinti di cui tre, in quanto clienti, vogliono acquistare tutti lo stesso prodotto e in quantità differente e il quarto è il gestore del magazzino che vuole aggiornare la quantità dello stesso prodotto acquistato dai clienti. Pur essendo le tipologie di utente differenti queste andranno comunque in concorrenza perché le operazioni sono tutte compatibili. Si mostra così che tutti i clienti non sono "bloccati" e possono procedere in concorrenza potendo effettuare al termine l'abort o il commit. Lo stato iniziale del sistema è pari alla quantità di prodotti disponibili che leggono tutti i clienti, mentre lo stato finale del sistema è la quantità di prodotti che rimangono. Se consideriamo la quantità iniziale di prodotto Qin=100 si avrà la seguente situazione:

Tipo utente	Tempi di commit/abort	Tempi di arrivo	Nome utente	codice prodotto	Qin	$Q_{acquisto}$ /Q_{resto} re	Q_{finale}	Prezzo_old	SQL	Prezzo_new
cliente	4+Δ	1	Saverio	1	100	1	100		R(Q) Q=Q-1	
cliente	4+2Δ	2	Francesco	1	100	2	100		R(Q) Q=Q-2	
cliente	4+3Δ	3	Nunzia	1	100	3	100		R(Q) Q=Q-3	
gestore	4+4Δ	4	Vincenzo	1	100	6	100		R(Q) Q=Q+6	

Tabella 1 – Caso d'uso con tutte operazioni compatibili

In tal caso la situazione iniziale del sistema è la seguente:

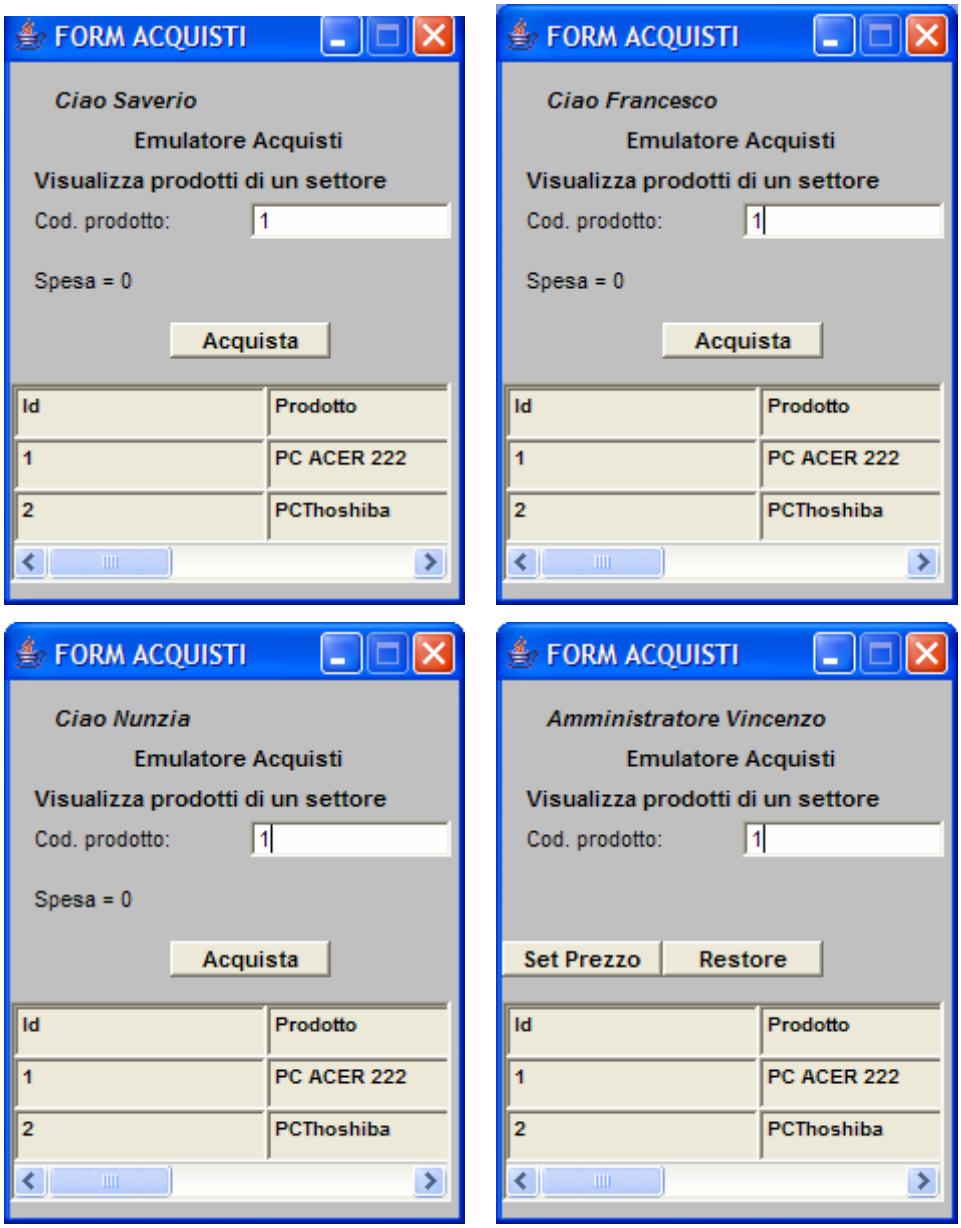


Figura 6 – Caso d'uso con quattro utenti di due tipologie e operazioni compatibili

La situazione finale è riportata nella figura seguente:

Utente connesso al tempo t>4+4 Δ

Figura 7 – Stato finale del caso d'uso con quattro utenti di due tipologie e operazioni compatibili

Nel secondo caso d'uso abbiamo quattro utenti: i primi due vogliono acquistare lo stesso prodotto, il terzo è il gestore del sistema che vuole settare il prezzo di quel prodotto del carrello e il quarto vuole effettuare l'acquisto del prodotto acquistato anche dai primi due clienti. Lo stato iniziale del sistema è pari alla quantità di prodotti disponibili che leggono tutti i clienti, mentre lo stato finale del sistema è la quantità di prodotti che rimangono. In questo caso si evidenzia che l'acquisto dell'ultimo cliente verrà fatto al nuovo prezzo settato dal gestore a differenza di quelli dei primi due clienti che avvengono in concorrenza al prezzo iniziale.

Tipo utente	Tempo di commit/abort	Tempo di arrivo	Nome utente	codice prodotto	Q_{in}	$Q_{acquisto}$ /$Q_{restore}$	Q_{finale}	SQL	Conto cliente	Prezzo_old	Prezzo_new
cliente	2+Δ	1	Saverio	1	100	1	94	R(Q) Q=Q-1	100	100	100
cliente	2+2Δ	2	Francesco	1	100	2	94	R(Q) Q=Q-2	200	100	100
gestore	3+Δ	3	Vincenzo	1	100	-	-	R(P) P=110	-	100	110
cliente	4+Δ	4	Nunzia	1	100	3	94	R(Q) Q=Q-3	330	100	110

Tabella 2 – Caso d'uso con tre operazioni compatibili e una incompatibile

La situazione iniziale è la seguente:

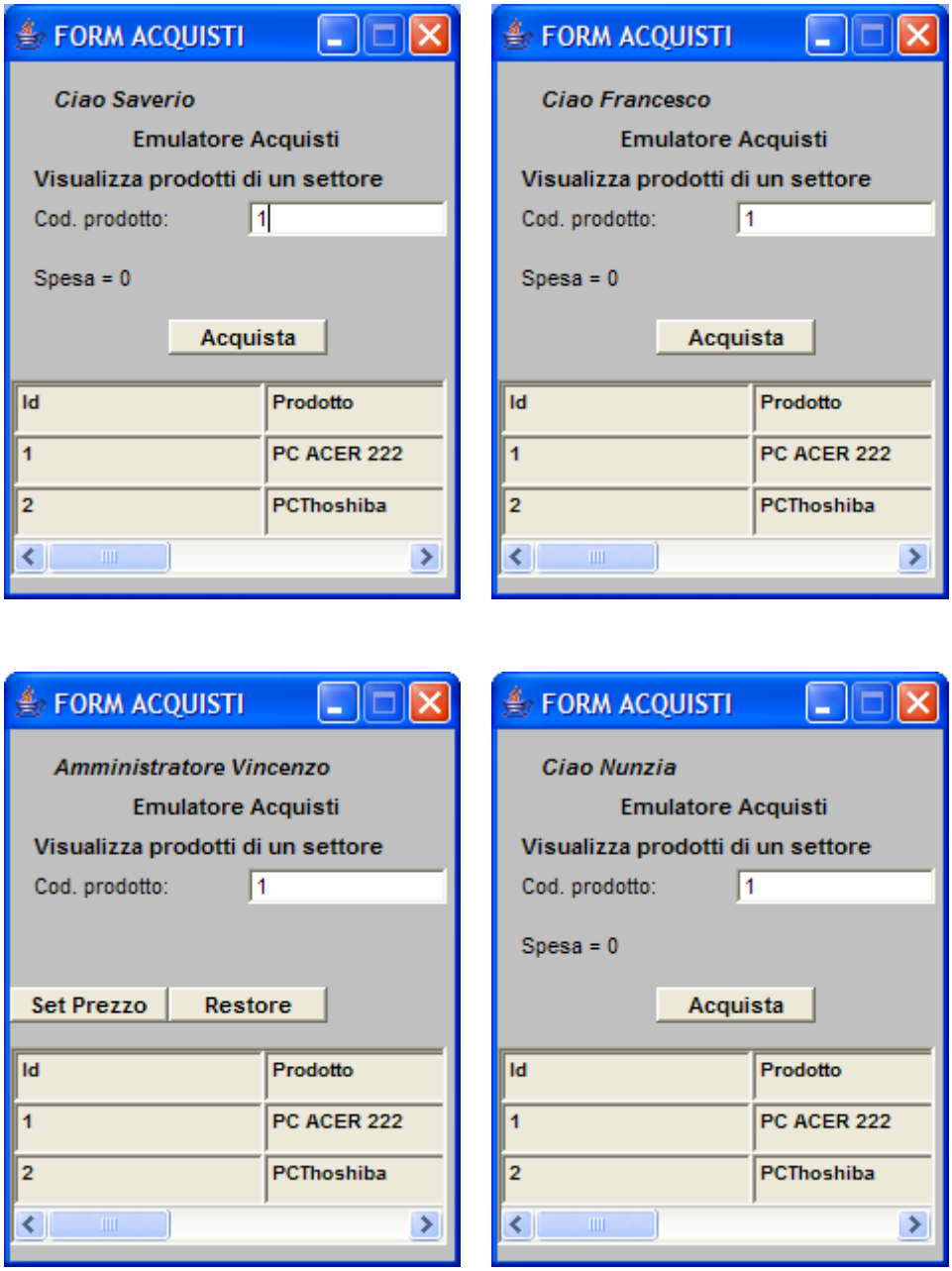


Figura 8 – Caso d'uso con quattro utenti di due tipologie e operazioni incompatibili

La situazione finale è la seguente:

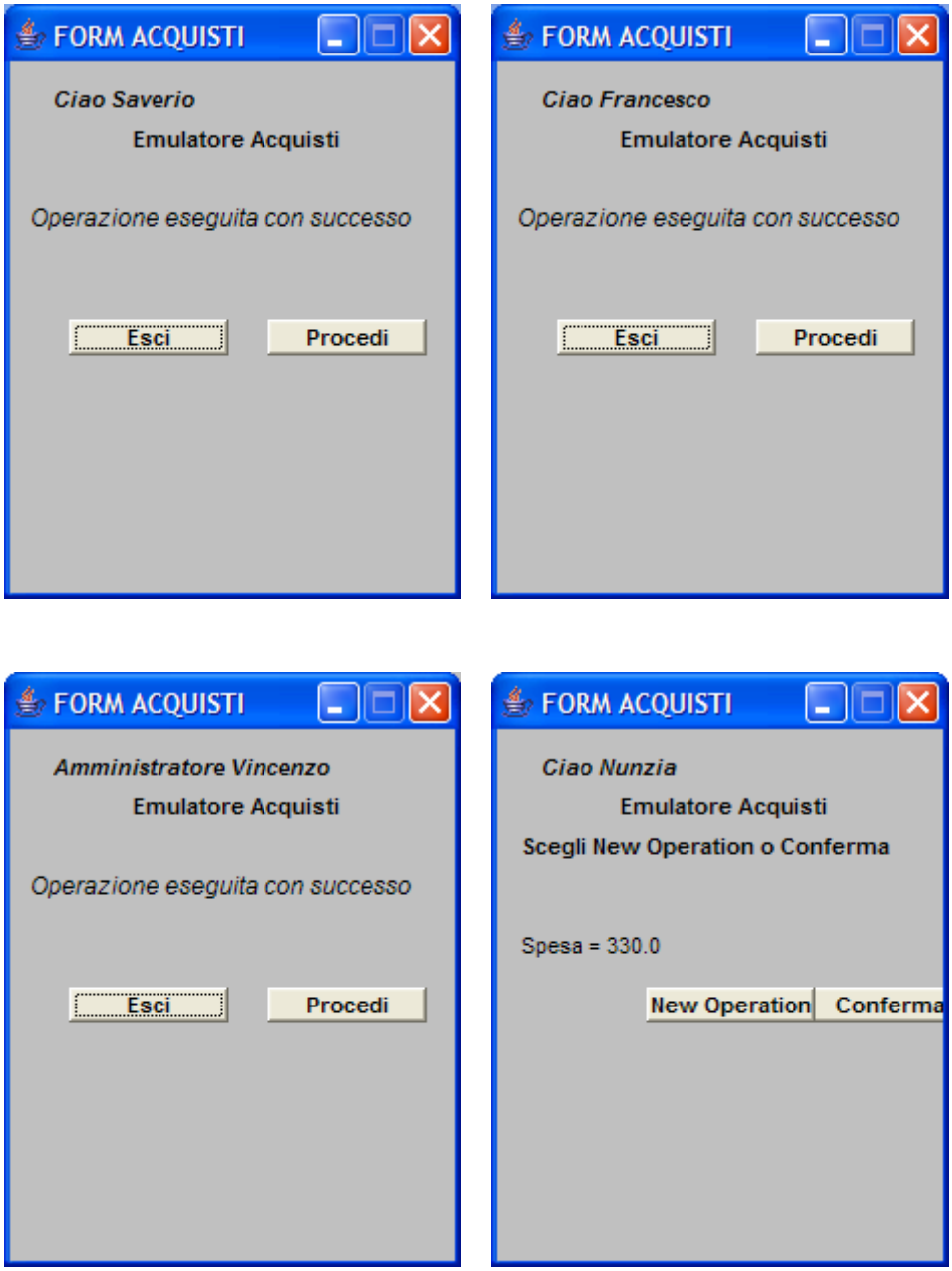


Figura 9 – Stato finale del caso d'uso con quattro utenti di due tipologie e operazioni incompatibili

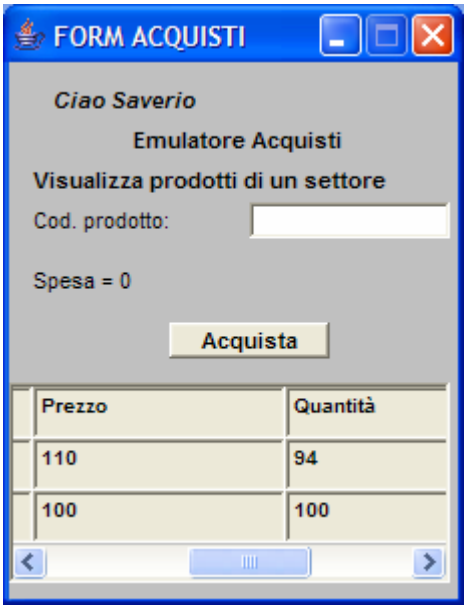


Figura 10 – Stato successivo al caso d'uso precedentemente mostrato per visualizzare l'aggiornamento su database

Nel terzo caso d'uso abbiamo sempre quattro utenti: i primi due vogliono acquistare lo stesso prodotto, ma il secondo si disconnette, il terzo è il gestore del sistema che vuole aggiornare il prezzo di quel prodotto del carrello e il quarto vuole acquistare un prodotto diverso da quello che hanno acquistato i primi due clienti. Quindi il primo e il secondo cliente agiscono in concorrenza ma per il secondo viene forzato l'abort perché la sua operazione a valle della disconnessione è seguita da un'operazione incompatibile. Procede quindi il gestore del magazzino aggiornando i prezzi e il quarto cliente acquista ai prezzi aggiornati.

Tipo utente	Tempo di commit	Tempo di arrivo	Nome utente	codice prodotto	Q_{in}	$Q_{acquisto}$ /$Q_{restore}$	Q_{finale}	SQL	Conto cliente	Prezzo_old	Prezzo_new
cliente	2+Δ	1	Saverio	1	100	1	99	R(Q1) Q1=Q1-1	100	100	100
cliente	3+2Δ	2	Francesco	1	100	2	99	R(Q1) Q1=Q1-2	0	100	100
gestore	3+Δ	3	Vincenzo	1	100			R(P) P=110		100	110
cliente	4+Δ	4	Nunzia	2	100	50	50	R(Q2) Q2=Q2-50	100	10	10

Tabella 3 – Caso d'uso con due operazioni compatibili, una compatibile disconnessa e una incompatibile

La situazione iniziale è la seguente:

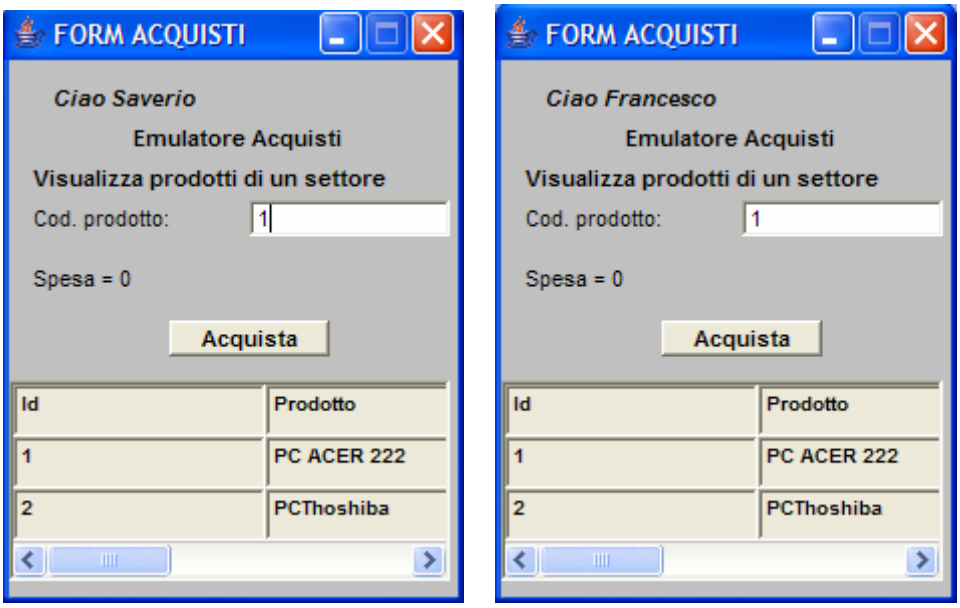
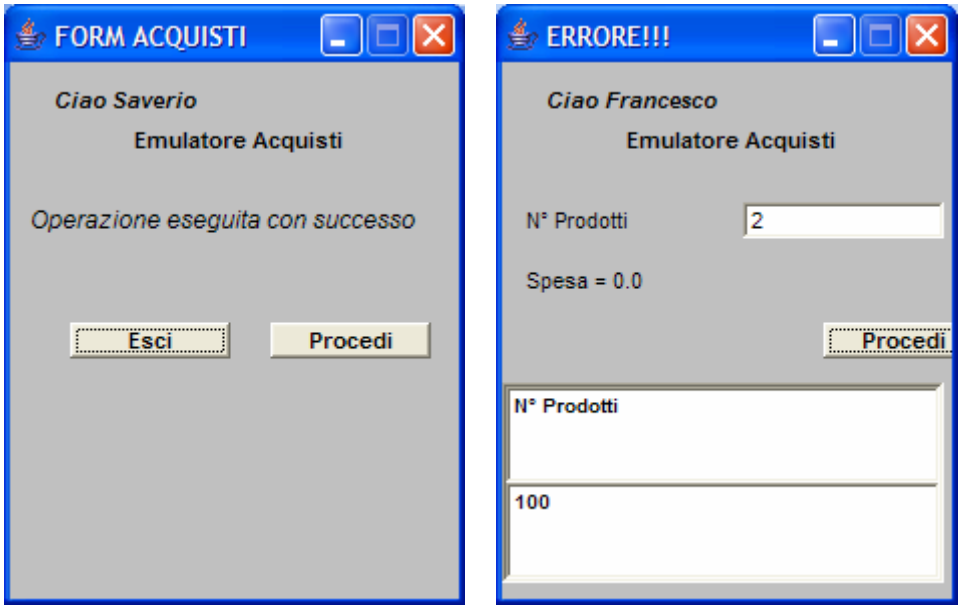


Figura 11 – Caso d'uso con quattro utenti di due tipologie e operazioni incompatibili con disconnessione

La situazione finale è la seguente:



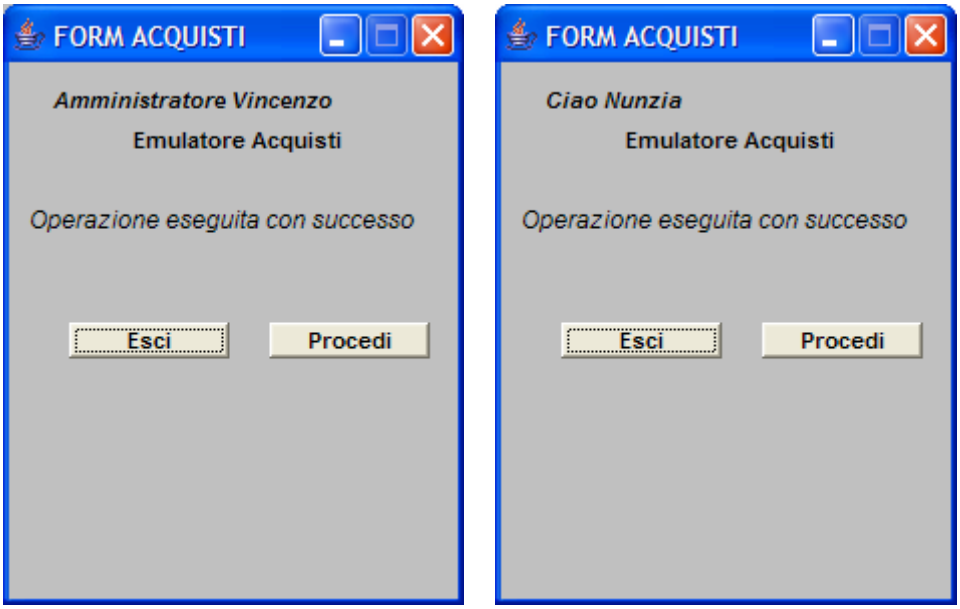


Figura 12 – Stato finale del caso d'uso con quattro utenti di due tipologie e operazioni incompatibili con disconnessione

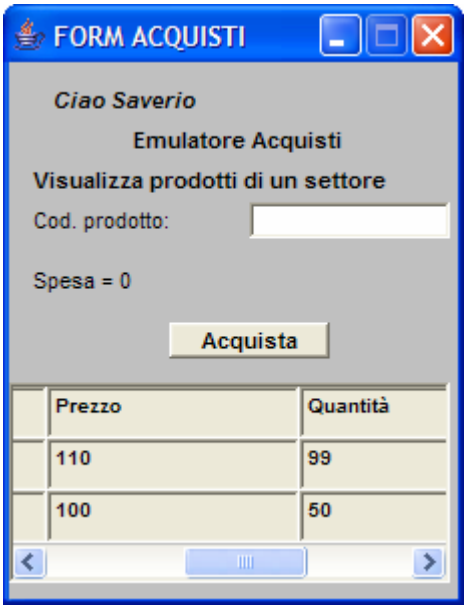


Figura 13 – Stato successivo al caso d'uso precedentemente mostrato per visualizzare l'aggiornamento su database

Nel quarto caso d'uso abbiamo quattro clienti: tutti e quattro vogliono effettuare acquisti di uno stesso prodotto, ma dei tre solo il secondo cliente si disconnette. C'è differenza rispetto al terzo caso perché oltre al fatto che il primo, il terzo e il

quarto cliente agiscono in concorrenza sulle medesime risorse, il secondo se si riconnette potrà portare a termine la transazione perché sulla risorsa hanno agito operazioni ad essa compatibili. Le quantità di prodotti disponibili al termine delle quattro transazioni sarà aggiornata correttamente.

Tipo utente	Tempo di Commit	Tempo di arrivo	Nome utente	codice prodotto	Q_{in}	$Q_{acquisto}$ /$Q_{restore}$	Q_{finale}	SQL	Conto cliente	Prezzo_old	Prezzo_new
cliente	2+Δ	1	Saverio	1	100	1	94	R(Q) Q=Q-1	100	100	
cliente	4+2Δ	2	Francesco	1	100	2	94	R(Q) Q=Q-2	200	100	
cliente	3+Δ	3	Vincenzo	1	100	1	94	R(Q) Q=Q-1	100	100	
cliente	4+Δ	4	Nunzia	1	100	2	94	R(Q) Q=Q-1	200	100	

Tabella 4 – Caso d'uso con quattro operazioni compatibili di cui una disconnessa

La situazione iniziale è la seguente:

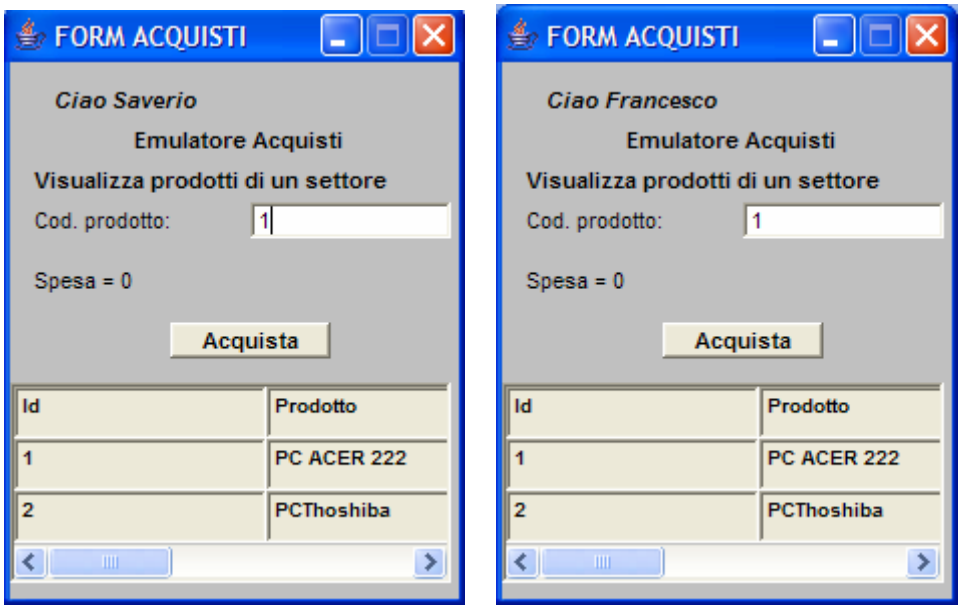


Figura 14 – Caso d'uso con quattro utenti di un'unica tipologia e operazioni compatibili

La situazione finale è la seguente:

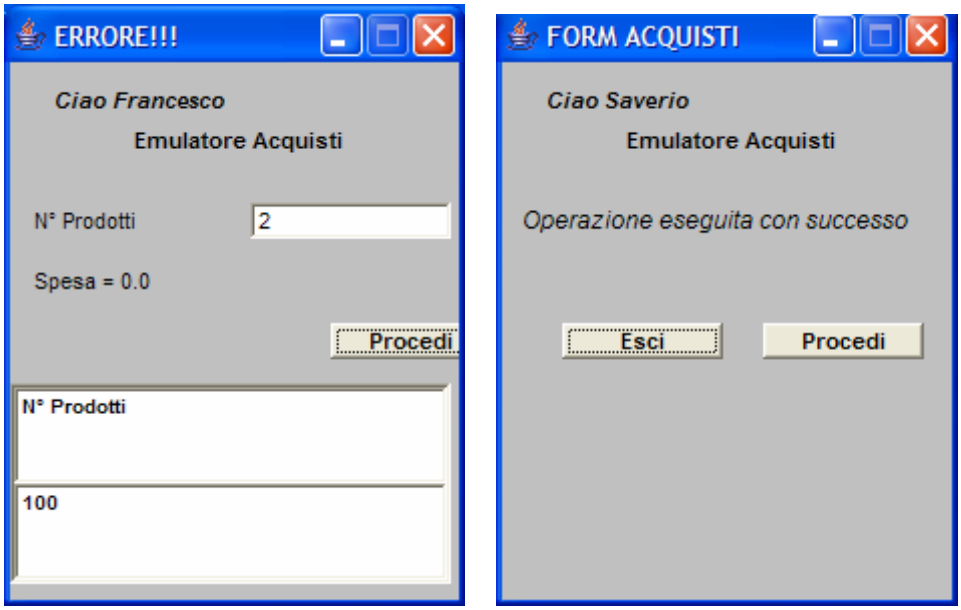


Figura 15 – Stato finale del caso d'uso con quattro utenti di un'unica tipologia e operazioni compatibili

Figura 16 – Stato successivo al caso d'uso precedentemente mostrato per visualizzare l'aggiornamento su database

Nel quinto caso d'uso la prima coppia di clienti vuole effettuare l'acquisto dello stesso prodotto mentre la seconda coppia vuole aggiornare il prezzo di alcuni

prodotti del carrello e supponiamo che i due gestori vogliano farlo per gli stessi prodotti. Quindi i primi due clienti acquisteranno i due prodotti in concorrenza e la quantità rimanente verrà correttamente aggiornata (ciò significa, e ciò vale per tutti i casi d'uso, che l'acquisto in concorrenza va a buon fine per il secondo cliente solo se la quantità di merce rimasta rispetta i vincoli di integrità referenziale della base dati). In caso contrario il secondo cliente vedrà forzato un abort al momento del commit.

Tipo utente	Tempo di commit	Tempo di arrivo	Nome utente	Codice prod.	Q_{in}	Qa / Qr	Qf	Conto cliente	Prezzo old	Prezzo new	Prezzo finale
cliente	2+Δ	1	Saverio	1	100	1	97	100			
cliente	2+2Δ	2	Francesco	1	100	2	97	200			
gestore	3+Δ	3	Vincenzo	2	100				100	150	200
gestore	4+Δ	4	Nunzia	2	100				100	200	200

Tabella 5 – Caso d'uso con due operazioni compatibili e 2 incompatibili

La situazione iniziale è la seguente:

Figura 17 – Caso d'uso con quattro utenti di due tipologie

La situazione finale è la seguente:

Figura 18 – Caso d'uso con quattro utenti di due tipologie

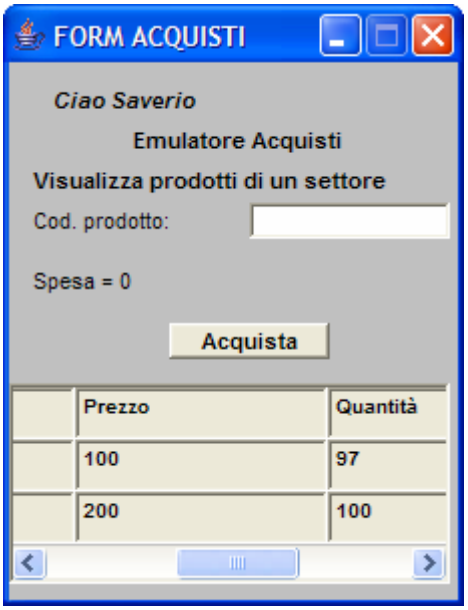


Figura 19 – Stato successivo al caso d'uso precedentemente mostrato per visualizzare l'aggiornamento su database

Come mostrato dai casi d'uso il sistema presenta nelle varie situazioni proposte per gli insiemi di input transazionali le stesse risposte attese teoricamente. Si dimostrano così la funzionalità e la correttezza del prototipo sviluppato.

4 Conclusioni e sviluppi futuri

La bontà dell'idea di introdurre la semantica delle operazioni delle transazioni per l'aumento della concorrenza delle stesse è stata avvalorata dall'analisi sperimentale condotta. Le analisi sui tempi di esecuzione hanno permesso di mettere in luce che l'applicazione dell'approccio proposto è particolarmente conveniente nel caso di percentuali medio alte di conflitti e percentuali basse di incompatibilità.
Quindi un ambiente ideale di applicazione del modello deve presentare un folto numero di operazioni delle transazioni della stessa natura. Per esempio il commercio elettronico (esempio al quale si ispira l'applicazione implementata) ha le caratteristiche citate visto che tutti gli acquisiti di merci a livello di query SQL non sono altro che delle semplici select finalizzate all'aggiornamento di quantità di una risorsa.
Si può poi pensare a sviluppi futuri del metodo utilizzato che si sintetizzano nei punti seguenti:

- Introduzione nel modello del concetto di priorità e rivisitazione delle politiche di gestione del deadlock e della starvation
- Ottimizzazione del middleware sviluppato ed integrazioni in un DBMS open source
- Confronto dell'approccio proposto con altri metodi presenti in letteratura su un dato data set standard

L'introduzione del concetto di priorità permetterebbe di non penalizzare eccessivamente alcuni tipi di transazioni che risultino sia poco frequenti nel sistema di schedulazione e sia incompatibili con un più folto numero di transazioni con operazioni compatibili. L'ottimizzazione del middleware sviluppato ovviamente permetterebbe anche un 'analisi dei tempi di esecuzione con data set standard. Questo perché l'overhead dell'applicazione così come concepita sicuramente non avrebbe permesso questo tipo di analisi.

Bibliografia e sitografia

[1] J. Jing, A.S. Helal, e A.K. Elmagarmid, "Client-server computing in mobile environments," ACM Computing Surveys, vol. 31, no. 2, 1999

[2] M. Satyanarayanan, "Mobile information access," IEEE Personal Communications, vol. 3, no. 1, 1996.

[3] M.H. Dunham e V. Kumar, "Location dependent data and its management in mobile databases," in Int. DEXA Workshop on Mobility in Databases and Distributed Systems, Vienna, Austria, Aug. 1998.

[4] M. Frodigh, P. Johansson, e P. Larsson, "Wireless ad hoc networking—The art of networking without a network," Ericsson Review, vol. 4, 2000.

[5] IETF, Mobile Ad-hoc Networks (manet), http://www.ietf.org/html.charters/manet-charter.html.

[6] A. Popovici e G. Alonso, "Ad-hoc transactions for mobile services," in VLDB Workshop on Technologiesfor E-Services, Hong-Kong, China, August 2002.

[7] J. Gray, P. Helland, P. O'Neil, e D. Shasha, "The dangers of replication and a solution," in ACM SIGMOD Conference, Montreal, Canada, June 1996.

[8] M. Lee e S. Helal, "HiCoMo: High Commit Mobile Transactions," Kluwer Academic Publishers Distributed and Parallel Databases (DAPD), vol. 11, no. 1, 2002.

[9] G.D. Walborn e P.K. Chrysanthis, "PRO-MOTION: Management of mobile transactions," in ACM Symp. on Applied Computing, San Jose, USA, March 1997.

[10] G.D.Walborn e P.K. Chrysanthis, "Transaction processing in PRO-MOTION," in ACM Symp. on Applied Computing, San Antonio, USA, Feb. 1999.

[11] P.K. Chrysanthis, "Transaction processing in a mobile computing environment," in IEEE Workshop on Advances in Parallel and Distributed Systems (APADS), Princeton, USA, Oct. 1993.

[12] S.K. Madria and B. Bhargava, "A transaction model for improving data availability in mobile computing," Kluwer Academic Publishers Distributed and Parallel Databases (DAPD), vol. 10, no. 2, 2001.

[13] L.H. Yeo e A. Zaslavsky, "Submission of transactions from mobile workstations in a cooperative multidatabase processing environment" in Int. Conf. on Distributed Computing Systems (ICDCS), Poznan, Poland,June 1994.

[14] K. Ku e Y. Kim, "Moflex transaction model for mobile heterogeneous multidatabase systems," In IEEE Workshop on Research Issues in Data Engineering, San Diego, USA, Feb. 2000.

[15] R.A. Dirckze e L. Gruenwald, "A toggle transaction management technique for mobile multidatabases," in Int. Conf. on Information and Knowledge Management (CIKM), Bethesda, USA, Nov. 1998.

[16] R.A. Dirckze e L. Gruenwald, "A pre-serialization transaction management technique for mobile multidatabases," Mobile Networks and Applications (MONET), vol. 5, no. 4, 2000.

[17] C. Bobineau, P. Pucheral, e M. Abdallah, "A unilateral commit protocol for mobile and disconnected computing," in Int. Conf. Parallel and Distributed Computing Systems (PDCS), Las Vegas, USA, Aug. 2000.

[18] V. Kumar, "A Timeout-based mobile transaction commitment protocol," in ADBIS-DASFAA Symp on Advances in Databases and Information Systems, volume 1884 of *LNCS*, Prague, Czech Republic,Sept. 2000.

[19] V. Kumar, N. Prabhu, M.H. Dunham, e A.Y. Seydim, "TCOT- A timeout-based mobile transaction commitment protocol," IEEE Transactions on Computers, vol. 51, no. 10, 2002.

[20] H. Garcia-Molina, "Using semantic knowledge for transaction processing in a distributed database," ACM Transactions on Database Systems (TODS), vol. 8, no. 2, 1983.

[21] T. Ozsu e P. Valduriez, Principles of Distributed Database Systems, Prentice Hall, 2nd ed., 1999.

[22] Y. Breitbart, H. Garcia-Molina, e A. Silberschatz, "Overview of multidatabase transaction management," Very Large Databases (VLDB) Journal, vol. 1, no. 2, 1992.

[23] H.T. Kung and J.T. Robinson, "On optimistic methods for concurrency control," in Int. Conf. on Very Large Databases (VLDB), Rio de Janeiro, Brazil, Oct. 1979.

[24] H. Berenson, P. Bernstein, J. Gray, J. Melton, E.J. O'Neil, e P.E. O'Neil, "A critique of ANSI SQL isolation levels," in ACM SIGMOD Conference, San Jose, USA, May 1995.

[25] K. Ramamritham e P.K. Chrysanthis, "A taxonomy of correctness criteria in database applications," Very Large Databases (VLDB) Journal, vol. 5, no. 1, 1996.

[26] K. Ramamritham e P.K. Chrysanthis, "Advances in concurrency control and transaction processing," IEEE Computer Society Press, 1996.

[27] J.J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," ACM Transactions on Computer Systems (TOCS), vol. 10, no. 1, 1992.

[28] L.B. Huston e P. Honeyman, "Peephole log Optimization," in IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, USA, Dec. 1994.

[29] M.H. Dunham e V. Kumar, "Defining location data dependency, transaction mobility and commitment," Technical Report 98-CSE-01, Southern Methodist University, Dallas, USA, Feb. 1998.

[30] P. Serrano-Alvarado, "Defining an adaptable mobile transaction service," in Int. EDBT Workshop on Young Researchers Workshop, number 2490 in *LNCS*, Prague, Czech Republic, March 2002.

[31] P. Serrano-Alvarado, C. Roncancio, M. Adiba, and C. Labb´e, "Environment awareness in adaptable mobile transactions," in Journes de Bases de Donnes Avances (BDA), Lyon, France, Oct. 2003.

[32] GuoQiong Liao, YunSheng Liu, LiNa Wang, ChuJi Peng "Concurrency Control of Real-Time Transactions with Disconnections in Mobile Computing Environment"

School of Computer Science & Technology, Huazhong University of Science& Technology (1998)

[33] Nitin Prabhu, Vijay Kumar Indrakshi Ray Gi-Chul Yang "Concurrency Control in Mobile Database Systems" SICE, Computer Networking Computer Science, Division of Information Engineering,University of Missouri Kansas City, Colorado State University ,Mokpo National University *(2001)*.

[34] Li Xiao-rong, Shi Bai-le "A Prediction-Based Approach to Concurrency Control Resulting in Low Blocking Rate under High-Quality Mobile

Environment" *Dept. of Computer and Information Technology, Fudan University (2005)*.

[35] Kam-Yiu Lam, Tei-Wei Kuo, Ben Kao, Tony S.H. Lee and Reynold Cheng **"**Evaluation of Concurrency Control Strategies for Mixed Soft Real-Time Database Systems" *(1999)*.

[36] Kam-yiu Lam, Tei-Wei Kuo, Wai-Hung Tsang and Gary C.K Law "Concurrency control in mobile distribuited real time database systems" Department of Computer Science City University of Hong Kong (Received December *1998*; in final revised form February *2000*).