UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA TRIENNALE

PARIDBMS: LOGIN

RELATORE: *Ch.mo Prof. Enoch Peserico Stecchini Negri De Salvi*

CORRELATORI: *Prof. Luca Pretto, Ing. Paolo Bertasi*

LAURENDO: *Alberto Rizzi*

A.A. 2009-2010

Sommario

Con la seguente tesi di laurea si intende presentare l'aggiornamento del plug-in DBMS, applicazione appartenente alla rete PariPari in via di sviluppo presso il Dipartimento di Ingegneria dell'Informazione dell'Università di Padova.
Tutte le nuove modifiche apportate al plug-in sono state effettuate sulla base del lavoro di creazione e sulle considerazioni future sviluppate nella prima versione di DBMS: ciò che si vuole ottenere è l'aggiunta di nuove funzionalità al plug-in e il miglioramento di alcune già presenti in termini di efficienza.
Dopo aver descritto la struttura della rete PariPari e del plug-in DBMS, creato nella sua prima versione dai colleghi Andrea Cecchinato, Jacopo Buriollo e Alessandro Costa, si procederà alla descrizione delle modifiche e aggiornamenti apportati al plug-in da me e dal mio collega Deniz Tartaro Dizmen.
Verranno spiegate le motivazioni che hanno portato a modificare il plug-in usufruendo di altri plug-in presenti nella rete PariPari come Connectivity e Diesel e a inserire nuove funzionalità come l'accesso remoto a un database e il problema ad esso connesso di potersi loggare al database a cui si vuole accedere.
In particolare la parte da me realizzata e che verrà descritta con maggior dettaglio riguarderà l'implementazione dell'integrazione di DBMS con Connectivity e di login.

Indice

1. Introduzione

Con questo elaborato si intende illustrare l'aggiornamento di Pari DBMS, esponendo gli aspetti significativi che caratterizzano il sistema e in particolare con maggior dettaglio le modifiche e le nuove funzionalità introdotte.
La progettazione e l'implementazione di Pari DBMS sono state realizzate nella loro prima versione dal gruppo di laureandi composto da Andrea Cecchinato, Jacopo Buriollo e Alessandro Costa, mentre l'aggiornamento di tale versione è stato progettato e implementato da me e dal laureando Deniz Tartaro Dizmen. Dunque per avere una visione in dettaglio delle funzionalità di base del sistema si consiglia la lettura delle tesi di laurea del gruppo dei tre laureandi, mentre per una visione più approfondita di una parte dell'aggiornamento si consiglia di leggere la tesi di laurea del mio collega.

Pari DBMS è un sistema di gestione di basi di dati memorizzate nella memoria di massa dei nodi connessi alla rete PariPari. Come qualsiasi altro DBMS (Database Management System) il sistema oltre alla gestione delle basi di dati deve garantire la loro consistenza, affidabilità, privatezza e offrire risposte efficaci ed efficienti a tutte le operazioni di interrogazione e aggiornamento che vengono effettuate. Nel garantire tali servizi vengono utilizzati dei meccanismi di funzionamento elaborati che tengono conto del fatto che il sistema è completamente decentralizzato e distribuito sui nodi di una rete peer-to-peer serverless: questa caratteristica impone al sistema di offrire l'ulteriore garanzia di disponibilità dei dati nella rete mantenendone la consistenza.

Prima di prendere in esame i meccanismi di funzionamento del sistema, si delinea una descrizione della rete PariPari e dei concetti fondamentali che la caratterizzano.

1.1 La rete PariPari

Come affermato precedentemente, Pari DBMS è a tutti gli effetti uno dei numerosi servizi offerti da PariPari, una nuova rete peer-to-peer serverless e multifunzionale.
Con il termine peer-to-peer si definisce un'architettura di rete in cui ogni nodo è in grado di svolgere sia il ruolo di client che di server verso tutti gli altri nodi della rete, offrendo e richiedendo allo stesso tempo risorse e servizi alla rete. Tali nodi si definiscono peer e si

distinguono dai nodi che costituiscono le reti con architettura client-server dove esiste una netta divisione dei ruoli tra nodi di tipo client che richiedono informazioni alla rete e nodi di tipo server che rispondono alle richieste provenienti dalla rete.
Tuttavia esistono reti peer-to-peer definite ibride poiché presentano al loro interno nodi privi del duplice ruolo che li caratterizza: è il caso di tutte quelle reti di peer come Napster, BitTorrent eDonkey2000 che sfruttano al loro interno dei veri e propri server per la ricerca di risorse o per instaurare una comunicazione tra i nodi. PariPari presenta invece una struttura completamente serverless, priva di nodi che fungono solamente da server e dalla cui disponibilità dipende il funzionamento della rete: una struttura cosi decentralizzata consente maggiore stabilità e garanzia di funzionamento alla rete ma comporta allo stesso modo una maggiore complessità nella sua gestione. In effetti PariPari consente ai propri peer di effettuare la ricerca delle risorse in modo affidabile ed efficiente grazie all'utilizzo di una tabella hash distribuita denominata DHT, nella quale ogni risorsa è rappresentata dall'indirizzo di rete del nodo che ne è responsabile e che risulta essere attivo in quel momento nella rete.

1.2 Multifunzionalità e struttura della rete PariPari

Nel paragrafo precedente è stata delineata la tipologia di rete che sta alla base del progetto PariPari ma non è stata trattata un'ulteriore e non meno importante caratteristica della rete stessa: la multifunzionalità.
PariPari si definisce multifunzionale in quanto è una rete in grado di gestire qualsiasi contenuto (file, pagine web, e-mail, stream, etc.) e di offrire numerosi servizi (VoIP, web server, IRC, DBMS, DNS, etc.) simili ai più comuni disponibili su internet. L'efficienza e l'affidabilità della gestione del gran numero di risorse e dei servizi offerti è resa possibile tramite una struttura modulare a plug-in. Ogni plug-in possiede compiti precisi e fornisce un determinato servizio direttamente all'utente o ad altri plug-in, affidando lo svolgimento di quelle funzioni che non gli competono agli altri moduli con i quali interagisce tramite il nucleo detto Core. Si tratta a tutti gli effetti di un approccio a scatola nera (black box) che permette a chi ha intenzione di usufruire di un modulo di passare le informazioni necessarie richieste dal modulo stesso e ottenere i risultati senza dover venire a conoscenza della sua realizzazione interna.
Tutti i plug-in non comunicano direttamente tra loro e vengono gestiti dal Core, il modulo centrale che ha il compito di inizializzarli e coordinarli tra loro: tutti i messaggi che i moduli si scambiano tra loro devono passare obbligatoriamente attraverso il Core che controlla che le richieste contenute possano essere eseguite e, in caso affermativo, le inoltra ai rispettivi

destinatari. Il Core non solo smista le richieste di risorse al corretto destinatario ma si occupa anche del percorso inverso, consegnando le risposte contenenti le risorse stesse al plug-in che le ha richieste inizialmente.

Oltre alla gestione dei plug-in tramite il Core, PariPari garantisce una gestione equa delle risorse e dei servizi messi a disposizione nella rete attraverso il sistema di gestione crediti. Tale sistema gode di un meccanismo che prevede che ogni risorsa o servizio della rete possieda un prezzo che ciascun nodo dovrà pagare per poterne usufruire, con lo svantaggio di dover rendere disponibili altre risorse da lui possedute. In questo modo ciascun nodo si rende partecipe nella rete in modo proporzionale alla quantità e qualità dei servizi richiesti.

I plug-in si dividono in due gruppi di appartenenza: la cosiddetta cerchia interna e cerchia esterna. Alla cerchia interna appartengono tutti quei moduli come DHT, Connectivity, Local Storage e Crediti, che si occupano della creazione e gestione della rete consentendone il corretto funzionamento. Alla cerchia esterna appartengono tutti quei moduli come DNS, VoIP, IRC; DBMS, che interagiscono con la cerchia interna per offrire un servizio di rete direttamente all'utente.

Pari DBMS, come appena affermato, appartiene alla cerchia esterna e sfrutta i moduli della cerchia interna per poter funzionare. Nella precedente progettazione si era previsto che il plug-in utilizzasse i moduli interni DHT, Connectivity e LocalStorage ma non se ne è tenuto conto totalmente nella sua prima realizzazione. L'unico modulo della cerchia interna utilizzato in questa prima versione è stato DHT, modulo che consente di gestire e ricercare le risorse nella rete e in particolare tutt'ora utilizzato dal plug-in per pubblicizzare e ricercare le basi di dati contenute nei nodi e l'eventuale disponibilità di un nodo a ospitare basi di dati. Connectivity e LocalStorage non sono stati presi in considerazione nella versione precedente del plug-in: Connectivity è il modulo che gestisce l'interfaccia di rete e fornisce socket a banda limitata a tutti i moduli che le richiedono, mentre LocalStorage è il modulo che permette la gestione della memoria di massa locale sfruttato dai plug-in per leggere o scrivere informazioni in memoria. Pari DBMS nella sua prima versione ha optato all'utilizzo di socket senza richiederle al modulo Connectivity, sfruttando una server socket per ricevere i messaggi dalla rete e diverse socket client per inviare i messaggi ai nodi. Attualmente nella nuova versione sviluppata, il plug-in richiede le socket di cui necessita a Connectivity consentendone inoltre una gestione più accurata che verrà trattata in dettaglio successivamente in questa tesi.

Per quanto riguarda LocalStorage sia la versione iniziale che quella attuale non lo utilizzano ma il plug-in sarà obbligato a farlo nelle prossime versioni che verranno rilasciate.

Tutt’ora il plug-in fa uso dei moduli DHT e Connectivity, ma usufruisce anche dell’ulteriore modulo Diesel che consente di distribuire un servizio sulla rete PariPari sfruttando la comunicazione tra i suoi nodi. Il servizio offerto da Diesel garantirà un miglioramento delle comunicazioni tra i nodi di PariPari riducendone la quantità. L’integrazione del plug-in con tale modulo verrà anch’essa trattata con maggior dettaglio successivamente nel seguito della tesi.

Quella che è stata descritta in questo paragrafo è la struttura della rete PariPari, applicazione realizzata attraverso un client scritto in Java. La scelta di tale linguaggio di programmazione è dettata dalla proprietà di portabilità che lo contraddistingue e che permette alle applicazioni di essere eseguite su diversi sistemi operativi senza doverle riadattare. Un qualunque utente che volesse accedere alla rete e ai servizi messi a disposizione deve semplicemente eseguire tale software su di un calcolatore connesso a Internet.

A questo punto termina la parte introduttiva. Nel prossimo capitolo si procederà alla descrizione di come il plug-in DBMS è stato realizzato nella sua prima versione esponendone la struttura interna e i meccanismi di funzionamento che ne stanno alla base.

2. Pari DBMS

Nel seguente capitolo verrà descritta la struttura interna di Pari DBMS, analizzando le diverse problematiche che hanno accompagnato la progettazione e realizzazione della prima versione del plug-in e esponendo brevemente le soluzioni che sono state adottate e quali metodi di funzionamento hanno determinato. Nei due capitoli successivi invece verranno analizzate le modifiche, in particolare le integrazioni del plug-in con i moduli Connectivity e Diesel, e le nuove funzionalità di accesso remoto e login introdotte rispetto a tale versione, anche se già previste, trattando dettagliatamente le problematiche che si sono presentate e i meccanismi di risoluzione adottati.

Prima di procedere con l'analisi delle problematiche, si fa presente che tutte le considerazioni che verranno effettuate in questo elaborato presuppongono l'utilizzo di un modello di dati relazionale. Inoltre nella realizzazione del progetto iniziale è stato scelto come sistema di gestione delle basi di dati il DBMS relazionale HSQLDB, evoluzione di Hypersonic SQL Project, scritto interamente in Java, in grado di supportare una vasta gamma di comandi SQL, interessante per la rapidità, stabilità e affidabilità offerta e usato particolarmente in progetti Open Source[1].

L'idea originale che sta alla base di Pari DBMS prevede che il plug-in realizzi un sistema di gestione di basi di dati distribuito in una rete peer-to-peer, dotato di prestazioni non troppo distanti da quelle che caratterizzano i DBMS tradizionali. Un sistema di questo tipo deve essere in grado di partizionare un database e distribuire le sue parti tra i nodi connessi alla rete, garantendo di potervi accedere da qualunque nodo e in qualunque momento. Il lavoro di ricerca che riguarda il partizionamento delle basi di dati in basi di dati di dimensioni inferiori è tutt'ora in corso di sviluppo: il sistema per il momento non è in grado di partizionare un database e di distribuire nella rete le varie parti prodotte, ma prevede che ogni base di dati venga replicata completamente in un certo numero di nodi.

Come tutti i DBMS tradizionali, anche un DBMS distribuito deve essere in grado di offrire consistenza e disponibilità dei dati che vengono gestiti, in modo efficiente ed efficace: in effetti più nodi nella rete possono contenere la stessa base di dati e una stessa transazione riguardante quella base di dati ed effettuata in un determinato momento su tali nodi dovrebbe produrre lo stesso risultato, uguale per tutti i nodi interessati. Risultati diversi si verificano nel caso che le basi di dati non siano tra loro allineate e dunque non contengano in un determinato momento lo stesso insieme di dati. Pari DBMS utilizza un meccanismo interno di sincronizzazione,

[1] Sito ufficiale del prodotto: http://hsqldb.org/

progettato e sviluppato nella prima versione del plug-in, che garantisce la consistenza di una base di dati presente nella rete, eseguendo nel corretto ordine le operazioni di aggiornamento richieste sulla base di dati e su tutte le sue replicazioni. Il sistema di sincronizzazione permette di garantire che tutte le basi di dati attive siano tra loro allineate in un determinato istante, ma non è in grado di prevedere l'allineamento di tutte quelle basi che non risultano attive nella rete in quel momento: infatti la rete peer-to-peer sui cui poggiano i servizi offerti da PariPari non garantisce una completa disponibilità dei nodi contenuti, che dunque possono disconnettersi casualmente per un tempo più o meno limitato e comportare la perdita del servizio di cui si necessita in quel lasso temporale. Nel caso di un DBMS distribuito, la disconnessione di un nodo dalla rete causerebbe la mancata disponibilità temporale di tutti i database in esso contenuto e il loro possibile disallineamento rispetto a tutte quelle repliche attive sulle quali possono essere eseguite, in quel periodo di tempo, operazioni di aggiornamento che andrebbero a modificare i dati contenuti internamente. Dunque per ovviare a questo problema, nella prima versione di Pari DBMS si è sviluppato un meccanismo di re-inizializzazione che permette a tutte quelle basi di dati che si sono disconnesse temporaneamente di potersi allineare con le loro repliche attive, non appena si riconnettono alla rete, tornando disponibili ad offrire quel servizio che garantivano precedentemente. Al meccanismo di re-inizializzazione se ne aggiunge un altro di fondamentale importanza nel funzionamento del plug-in: il meccanismo di replicazione a caldo che garantisce che non avvenga la disconnessione di tutti i nodi contenenti un particolare database. Tale sistema consente di monitorare il numero di replicazioni attive riguardanti un database, andando a conoscere quali sono i nodi momentaneamente attivi che lo contengono e ovviamente il loro numero. Non appena il numero diminuisce al di sotto di una certa soglia prestabilita, si avviano dei processi di replicazione che creano delle vere e proprie copie di tale database nella rete. In questo modo la disponibilità di un determinato database non viene mai a mancare.

Tutte le problematiche appena descritte e le soluzioni adottate verranno trattate con maggior dettaglio nel seguito di questo capitolo, mentre per il momento verranno esposti quei problemi già messi in luce nella progettazione della prima versione del plug-in ma che non sono stati presi in considerazione inizialmente.

Come accennato nel capitolo precedente, nella prima progettazione di Pari DBMS si era deciso che il plug-in dovesse sfruttare il servizio offerto dal modulo Connectivity per poter interagire con la rete. Connectivity è in grado di offrire socket a banda limitata, utilizzate per la comunicazione reciproca tra i nodi della rete. Nella prima realizzazione infatti ogni nodo non effettuava richieste al modulo Connectivity per recuperare le socket necessarie per inviare e

ricevere i messaggi, ma era dotato internamente di una server socket con cui riceveva i messaggi dalla rete e di un numero vario di socket con cui inviava i messaggi nella rete. In particolare per ogni messaggio inviato il nodo utilizzava una socket diversa, comportando lo spreco di un gran numero di socket nel caso in cui doveva inviare molti messaggi. Tuttora il plug-in prevede l'utilizzo di Connectivity e di un meccanismo di gestione delle socket sfruttate per l'invio dei messaggi che evita lo spreco di tali risorse consentendone il riutilizzo. L'integrazione con il modulo Connectivity e la progettazione e realizzazione del meccanismo di gestione delle socket verranno descritti con maggior dettaglio nel prossimo capitolo. Inoltre in questa nuova versione di Pari DBMS si è deciso di effettuare l'integrazione con il Diesel: tale modulo consente ai nodi in cui è attivo il plug-in di comunicare tra loro riducendo il numero di messaggi scambiati. Prima dell'integrazione con Diesel ogni nodo doveva inviare ripetutamente una serie di messaggi ai nodi con cui aveva in comune un servizio in modo da comprendere quando era avvenuta una loro disconnessione dalla rete: questo meccanismo comportava uno spreco inutile di messaggi. Diesel permette ai nodi di scambiarsi informazioni e di venire a conoscenza della loro disconnessione attraverso l'utilizzo di una quantità minore di messaggi, migliorando così l'efficienza del sistema stesso. L'integrazione con il modulo Diesel verrà descritta anch'essa con maggior dettaglio nel prossimo capitolo.

Una ulteriore problematica riguarda la connessione di un utente ad una base di dati situato in un nodo che non la contiene direttamente. Nella prima versione del plug-in un utente era in grado di accedere a una particolare basi di dati solamente se si trovava nel nodo che la conteneva direttamente. Tuttora un utente può accedere a una base di dati da qualunque nodo della rete che usufruisca del servizio Pari DBMS: tramite il meccanismo di gestione dell'accesso remoto un utente può effettuare qualunque tipo di interrogazione da remoto e riceverne i risultati. Tuttavia permettere a qualunque utente di accedere a un particolare database non sarebbe del tutto corretto, in quanto alcuni utenti potrebbero manipolare i dati contenuti creando inconsistenza: l'accesso a una base di dati deve essere controllato sfruttando delle credenziali di accesso in modo che solo gli utenti fidati possano accedervi e apportare modifiche. Dunque il meccanismo di gestione dell'accesso remoto utilizza al proprio interno un meccanismo di login con cui si verifica se l'utente che ha richiesto di accedere a un dato database possa effettivamente connettersi. Il controllo delle credenziali non deve avvenire solamente nel caso un utente voglia connettersi a un database da remoto ma anche nel caso che voglia connettersi a un database contenuto localmente.

La progettazione e realizzazione dei meccanismi che riguardano l'accesso remoto e il login verranno trattati dettagliatamente nel quarto capitolo riguardante le nuove funzionalità introdotte in Pari DBMS.

2.1 Partizionamento

All'inizio di questo capitolo si è accennato al fatto che Pari DBMS dovrebbe essere in grado di partizionare le basi di dati e di distribuire le varie parti tra i nodi della rete, garantendo il corretto funzionamento delle basi stesse. Tramite il partizionamento, ogni nodo non sarebbe costretto a memorizzare completamente una base di dati, ma solo una sua piccola parte, riducendone la quantità di memoria associata. In tal modo la massima estensione di una base di dati non sarebbe limitata alla capacità di memorizzazione del nodo che la conterrebbe.
Inoltre il meccanismo di replicazione non creerebbe nei nodi repliche di intere basi di dati ma solamente di frammenti, occupandone meno memoria e rendendo più rapido il processo. Le risorse che i nodi condividerebbero tra loro non sarebbero più intere basi di dati ma i loro frammenti o lo spazio di memorizzazione in cui salvare nuovi frammenti.
Tuttavia questo meccanismo risulta essere molto complesso e comporterebbe a sua volta delle problematiche. La prima riguarderebbe la modalità di partizionamento di una base di dati: bisogna decidere con quali regole partizionare la base considerando che il suo schema può andare incontro a modifiche nel tempo. Inoltre si deve considerare anche la possibilità che una singola parte potrebbe aumentare a tal punto da dover ricorrere a un suo partizionamento in più parti oppure potrebbe rimanere vuota e dunque occupare risorse inutilmente. Un'altra problematica riguarderebbe il carico di lavoro di tutti quei nodi che riceverebbero operazioni riguardanti database di cui conterrebbero un frammento: il partizionamento comporterebbe per tali nodi un carico di lavoro maggiore, in quanto dovrebbero ricostruire l'intero schema relazionale della base di dati interrogata e successivamente dovrebbero suddividere le richieste di lettura e aggiornamento in sotto-richieste da inviare alle partizioni interessate.
Un'ulteriore problematica riguarderebbe la gestione di basi di dati molto estese: il numero di frammenti prodotti di piccole dimensioni sarebbe elevato, come il numero di messaggi che verrebbero scambiati in rete per eseguire le operazioni su di esse e nel caso di aggiornamenti per poter sincronizzare le varie replicazioni dei frammenti modificati dalle operazioni. Inoltre l'esecuzione delle operazioni potrebbe compromettere l'efficienza del sistema, comportando molto dispendio di tempo, dovuto all'eventuale recupero dei frammenti da cui reperire le

informazioni richieste e alla ricostruzione della porzione di database necessaria per eseguire l'interrogazione.
La risoluzione di questi problemi non è semplice e tutt'ora il sistema non gode del meccanismo di partizionamento in quanto ogni calcolatore contiene al proprio interno basi di dati complete.
Attualmente è in corso uno studio di analisi dei vari meccanismi di partizionamento esistenti in modo da determinare il migliore per le basi di dati distribuite e che ne consenta l'accesso simultaneo. In futuro dunque si prevede di dotare il plug-in di tale meccanismo in modo da usufruire di tutti i vantaggi che comporterà questa scelta.

2.2 Sincronizzazione e allineamento

Precedentemente è stato introdotto il problema di mantenere una base di dati e le sue replicazioni allineate tra loro in modo da garantire la consistenza dei dati stessi. Il meccanismo che lo consente viene definito processo di sincronizzazione e consiste nel far eseguire a una base di dati e alle sue replicazioni le stesse operazioni nel corretto ordine cronologico.
Per poter garantire la sincronizzazione il sistema si avvale dell'utilizzo di identificatori temporali (o timestamp): ad ogni nodo è associata una variabile intera che rappresenta il valore del suo timestamp locale; ad ogni operazione, nel momento della creazione, viene associato un identificatore temporale che ne permette l'ordinamento. Utilizzando una variante dell'algoritmo a timestamp distribuiti di Lamport[2], il sistema garantisce il corretto ordine nell'esecuzione delle operazioni in tutti i nodi interessati dalla sincronizzazione. Ogni nodo mantiene sincronizzato il proprio timestamp locale con quello degli altri calcolatori della rete: quando un messaggio viene inviato nella rete, il nodo mittente ne associa il proprio timestamp locale, il quale viene recuperato dal destinatario, incrementato e successivamente confrontato con il proprio timestamp locale. Se il timestamp giunto dal mittente e incrementato è maggiore del timestamp locale, il destinatario aggiorna la propria variabile temporale al nuovo valore ricevuto, altrimenti la aggiorna al valore precedente incrementato di una unità.
Inoltre ogni nodo nel momento di creazione di un'operazione, come detto precedentemente, recupera il valore attuale del proprio timestamp locale, lo incrementa e lo associa all'operazione creata, sia nel caso si tratti di un'operazione di reperimento che di aggiornamento. Il processo di sincronizzazione tuttavia riguarda solamente le operazioni di aggiornamento, che devono essere eseguite in tutte le repliche della base di dati interessata in modo da mantenere i dati allineati tra loro. Le operazioni di recupero invece possono essere compiute consultando una sola copia

[2] Leslie Lamport, noto ricercatore statunitense dal quale deriva il nome dell'algoritmo.

della base, in quanto attraverso la sincronizzazione la base e le sue repliche in un determinato istante conterrebbero gli stessi dati e una loro interrogazione molteplice produrrebbe lo stesso risultato risultando inutile e diminuendo l'efficienza del sistema.
Nel momento in cui un nodo riceve la richiesta di esecuzione di un'operazione di aggiornamento, esso deve creare l'operazione e prima di procedere con la sua esecuzione deve mettersi in contatto con tutti quei nodi contenenti una replica della base di dati da aggiornare, in modo da verificarne la possibilità di esecuzione. Il nodo invia a tutti i nodi interessati dall'aggiornamento un messaggio in cui richiede l'esecuzione locale di tale operazione e successivamente si mette in attesa delle risposte. Ogni nodo che riceve tale richiesta, recupera l'operazione in essa contenuta e verifica se può effettivamente eseguire tale operazione valutandone il timestamp associato e controllando di non aver già eseguito localmente un'operazione con timestamp di valore maggiore o uguale. Se ciò avviene, anche per una sola replicazione, significa che l'operazione è già stata compiuta per tale base di dati e che il nodo che ha effettuato la richiesta possiede un database non allineato con le proprie repliche. Dunque il processo di sincronizzazione deve essere interrotto dal mittente della richiesta stessa in modo da evitare che tale operazione venga eseguita sia in locale che nelle replicazioni: ad ogni nodo verrà inviato un messaggio che vieti l'esecuzione locale dell'operazione. Nel caso contrario, invece, quando tutti i nodi contenenti le replicazioni hanno approvato l'esecuzione dell'aggiornamento, il nodo mittente inserisce l'operazione in una coda a priorità ordinata secondo timestamp crescente, dove attende di essere eseguita, e invita gli altri nodi a compiere la stessa procedura inviando un messaggio che conferma l'esecuzione dell'operazione stessa.
L'estrazione dell'operazione dalla coda e la sua successiva esecuzione avvengono solo nel caso che la base di dati interrogata si trovi nello stato Ready. Nel caso il suo stato sia a Paused, la base continua a partecipare al processo di sincronizzazione di eventuali nuovi aggiornamenti che la interessano, approvandone o bocciandone l'esecuzione, ma si limita a inserirli nella coda a priorità bloccandone l'esecuzione che verrà ripresa una volta che lo stato ritorna a Ready.

Per una visione più approfondita della sincronizzazione delle operazioni in Pari DBMS si consiglia di consultare la tesi di Alessandro Costa.

2.3 Re inizializzazione

Uno dei problemi affrontati all'inizio di questo capitolo riguarda la disconnessione dei nodi dalla rete e ciò che comporta per il plug-in. Quando un nodo si disconnette, tutte le risorse che

sono contenute all'interno della sua memoria di massa, come le basi di dati o lo spazio a disposizione per salvare nuovi database, non sono più raggiungibili. Dunque tutti i messaggi e tutte le operazioni destinate ai database di tale nodo non possono raggiungere il nodo: in particolare il problema consiste negli aggiornamenti riguardanti le basi di dati contenute che non possono essere compiuti durante tutto l'arco di inattività del nodo rendendo le basi disallineate rispetto a quelle contenute nei nodi attivi che invece sono rimaste allineate grazie al processo di sincronizzazione. Quando un nodo disconnesso si riconnette alla rete potrebbe contenere dati non aggiornati e dunque potrebbe necessitare di un meccanismo che gli permetta di aggiornarli: il processo di re inizializzazione dei dati. Tale processo consente a tutte quelle basi che per qualche motivo si sono disallineate rispetto alle corrispondenti replicazioni di potersi riallineare e diventare nuovamente utilizzabili dagli utenti e dalle applicazioni.

Il meccanismo funziona nel seguente modo. Quando un nodo si riconnette imposta lo stato delle basi di dati da aggiornare a Updating e fa partire per ognuna di queste un processo di allineamento: ogni singolo processo si occupa di ricercare nella rete una copia della base di dati per cui è stato lanciato, posta allo stato Ready, a cui connettersi per recuperare tutte le operazioni di aggiornamento che sono state eseguite nel periodo di disconnessione. Tali operazioni vengono poi eseguite localmente in modo da allineare la base di dati non aggiornata. La base di dati da cui vengono recuperate le operazioni viene posta anch'essa allo stato Update per tutto il processo di allineamento, terminato il quale tornerà allo stato Ready. Le operazioni di modifica che vengono eseguite su qualunque base di dati in un qualsiasi calcolatore, vengono memorizzate all'interno di un'unica lista comune a tutti le basi locali, detta anche log delle transazioni, nella quale non vengono salvate solamente le query SQL contenute ma anche altre informazioni associate, come il nome della base di dati su cui è stata eseguita e il timestamp associato all'operazione stessa. All'inizio del processo di allineamento di una base di dati non aggiornata, il nodo che la contiene invia al nodo stabilito, contenente la copia aggiornata di tale base, una richiesta di aggiornamento associandone il timestamp dell'ultima operazione eseguita per tale base di dati, definito come lastTimeStamp (lts). Il nodo che riceve la richiesta recupera a sua volta il timestamp dell'ultima operazione eseguita dal proprio log delle transazioni, definito maximumTimeStamp (maxRTS), e successivamente effettua un confronto tra i due timestamp ottenuti. Nel caso che il timestamp ricevuto sia maggiore o uguale a quello ricavato dal proprio log delle transazioni (lts > = maxRTS), il processo di allineamento termina in quanto la base di dati che risultava non aggiornata in realtà risulta allineata a tutti gli effetti. Nel caso invece che tale quantità sia minore di quella ricavata localmente (lts < maxRTS), allora il processo di allineamento continua: il nodo remoto recupera dal log delle transazioni tutte le operazioni

riguardanti la base di dati interessata dal processo di allineamento, in particolare tutte quelle che hanno identificatore temporale maggiore rispetto a quello ricevuto, e le invia al nodo richiedente in modo che possa eseguirle per attualizzarsi. Tuttavia per questo caso potrebbe verificarsi un problema dovuto al fatto che il log delle transazioni è limitato e non può contenere tutte le operazioni che vengono eseguite nei database locali: in effetti secondo dei metodi ben definiti, periodicamente vengono eliminate le operazioni eseguite con timestamp al di sotto di una certa soglia che viene incrementata di volta in volta. Dunque quando il nodo remoto riceve la richiesta di aggiornamento per una particolare base di dati, esso deve recuperare dal proprio log delle transazioni non solo il timestamp dell'ultima operazione eseguita per tale base di dati (maxRTS), ma anche il timestamp della prima operazione associata a tale base e contenuta in tale lista in quell'istante, definito come minimumTimeStamp (minRTS). Se il timestamp ricevuto è compreso internamente a questi due valori (minRTS <= lts < maxRTS) allora si procede con il recupero delle operazioni, altrimenti se il timestamp risulta inferiore anche del timestamp più basso (lts < minRTS) il processo di allineamento viene terminato in quanto la base di dati che deve essere aggiornata contiene una versione così obsoleta dei dati da non riuscire a recuperare in rete tutte le operazioni di modifica necessarie a riallinearla. Il nodo che la contiene è costretto a eliminare tale base dalla memoria locale.

La procedura di allineamento dopo aver ottenuto le operazioni già eseguite dal log delle transazioni remoto, deve recuperare anche tutte quelle operazioni contenute nella coda delle transazioni in remoto associata al database da allineare e già confermate. Il nodo remoto invia al nodo richiedente l'aggiornamento dei messaggi contenenti l'insieme di operazioni che una volta ricevute, vengono poi inserite nella coda di transazioni locale per essere eseguite successivamente. Terminato l'invio di tutte le operazioni il nodo remoto ritorna allo stato Ready.

Durante il processo di allineamento entrambi i nodi interessati partecipano al processo di sincronizzazione delle nuove operazioni di modifica, inserendole nelle proprie code di transazioni rispettivamente ma attendendo di eseguirle. Nel momento in cui il nodo richiedente ha eseguito tutte quelle operazioni che erano state ricavate dal log delle transazioni remoto e dunque risulta allineato rispetto alle proprie repliche, pone il proprio stato a Ready e a questo punto potrà procedere con l'esecuzione delle operazioni rimaste nella coda delle transazioni, ritornando a funzionare normalmente come avveniva prima della disconnessione.

Maggiori dettagli sul processo di allineamento dei dati sono reperibili nell'elaborato di Jacopo Buriollo.

2.4 Replicazione

L'ultimo problema trattato riguarda l'indisponibilità di una base di dati. Uno degli obiettivi principali di Pari DBMS è quello di garantire in ogni istante la disponibilità delle basi di dati create in modo da poter risultare sempre utilizzabili dagli utenti e dalle applicazioni. Questa proprietà viene a mancare nel momento in cui tutti i nodi che contengono una particolare base si disconnettono dalla rete. Per evitare che ciò si verifichi il modulo gode del meccanismo di replicazione a caldo in grado di controllare il numero delle replicazioni attive di una particolare base di dati e di crearne nuove copie nel momento questo si abbassi al di sotto di una certa soglia. Come è stato detto in precedenza ogni nodo è in grado di pubblicizzare le risorse contenute, come le proprie basi di dati o lo spazio di memoria messo a disposizione, attraverso il modulo DHT, permettendone in questo modo anche la ricerca. Ogni nodo può in qualunque momento ricercare tramite DHT quali altri nodi contengono una copia di una particolare base di dati da lui posseduta, senza sapere se tali nodi siano attualmente connessi o disconnessi alla rete. Tuttavia il nodo può conoscere se tali nodi sono attivi inviando ad essi un messaggio di ping e attendendo la risposta: se entro un certo timeout la risposta non giunge al mittente allora il nodo contattato viene considerato disconnesso dalla rete. I nodi possono così memorizzare all'interno di più liste, una per ogni base di dati locale, tutti quei nodi attivi che ne possiedono una copia mantenendo le liste di volta in volta aggiornate e monitorate. Nel momento in cui il numero di copie disponibili di un particolare database scende al di sotto di una soglia prestabilita, ciascuna di esse deve lanciare un processo di replicazione con una certa probabilità p. L'esistenza di questo parametro è dovuta al fatto che non tutte le copie attive devono lanciare il processo di replicazione poiché tale meccanismo non permette loro di poter eseguire nessuna operazione. Se ciò accadesse allora si verificherebbe nuovamente un problema di indisponibilità dovuto in questo caso alla mancata possibilità di esecuzione delle operazioni in quanto tutte le copie di un particolare database sarebbero impegnate nel processo di replicazione. In questa maniera solo alcune copie sono interessate dal processo di replicazione mentre le rimanenti sono disponibili all'esecuzione delle operazioni. Il sistema utilizza come soglia di replicazione un numero di nodi attivi pari a 12 al di sotto del quale il sistema comincia a replicare e come parametro p di probabilità di replicazione il valore 0,7. Per distinguere le copie interessate dalla replicazione rispetto a quelle disponibili per l'esecuzione delle operazioni, il sistema stabilisce lo stato delle prime a Paused e delle seconde a Ready: allo stato Paused un database non è in grado di eseguire le operazioni che gli giungono o che contiene nella propria coda delle transazioni, ma si limita solamente a partecipare al processo di sincronizzazione aggiungendo nella coda le operazioni di modifica che lo riguardano e eseguendole una volta tornato allo stato Ready, ossia

quando il processo di replicazione è terminato, in modo da potersi riallineare con le copie disponibili.

Il processo di replicazione si svolge in due fasi: la prima fase consiste nella ricerca tramite DHT di un nodo che abbia disponibilità a ospitare nuovi database e in particolare che possa contenere tale copia dei dati. Infatti esistono due regole precise a cui devono attenersi i calcolatori riguardanti la memorizzazione di un database al loro interno: la prima afferma che un calcolatore non può contenere al suo interno un numero di database superiore a un numero massimo, stabilito a 5; la seconda afferma che un calcolatore non può contenere al suo interno due o più copie dello stesso database. La fase di ricerca termina non appena viene trovato un nodo idoneo ad ospitare la base da replicare.

La seconda fase consiste nell'invio di alcuni messaggi al nodo ottenuto dalla ricerca precedente: si tratta di messaggi di replicazione che contengono informazioni riguardanti la base di dati da creare e i dati che sono contenuti al suo interno al momento del lancio della replicazione. Il nodo che si occupa della replicazione recupera le informazioni riguardanti lo schema logico-relazionale della base da replicare e le invia al nodo destinatario della replicazione che a sua volta sulla base dei messaggi ricevuti è in grado di ricostruire il database. Successivamente il nodo mittente recupera l'insieme dei dati contenuti nella base di dati da replicare e li invia tramite più messaggi al destinatario che li utilizza per popolare la base di dati appena creata.

I messaggi al loro interno contengono query SQL a tutti gli effetti, per la creazione delle tabelle e per l'inserimento dei dati in esse. Al termine del processo di replicazione la nuova copia prodotta risulterà disallineata rispetto alle altre copie attive e dunque si allineerà utilizzando la procedura di aggiornamento descritta precedentemente.

Maggiori dettagli sulla replicazione si possono trovare nell'elaborato di Andrea Cecchinato.

2.5 Architettura logica

Il modulo Pari DBMS è organizzato in ogni nodo tramite una struttura logica a tre strati sovrapposti: la memoria di massa locale, un DBMS tradizionale e infine il gestore del nodo.

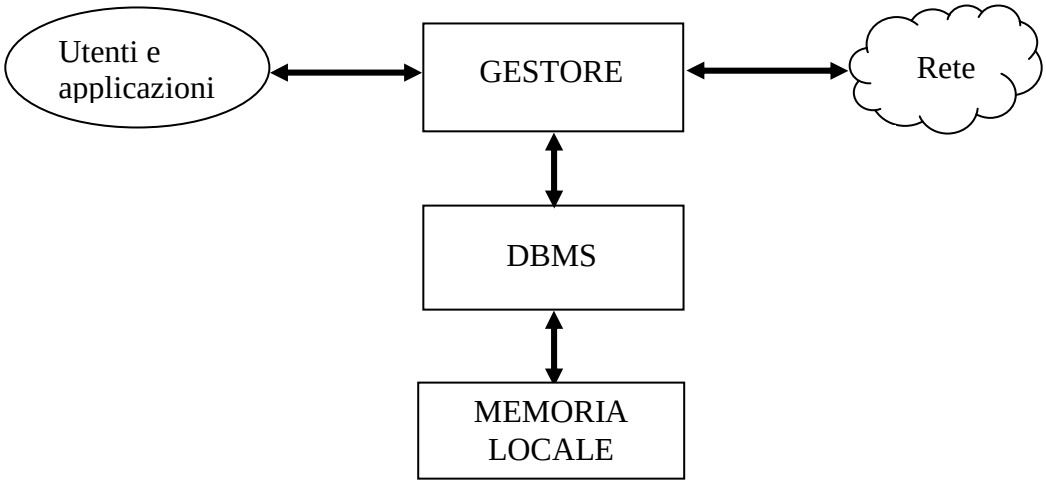


Figura 2.1: Schema logico di Pari DBMS

Gli utenti o le applicazioni che vogliono usufruire del servizio offerto dal plug-in, affidano l'esecuzione delle query SQL al gestore del nodo, il quale è in grado di interagire direttamente con la rete, e dunque con i gestori degli altri nodi connessi, oppure con il livello logico sottostante corrispondente al DBMS locale. Una volta ricevuta la query, il gestore locale la smista al DBMS sottostante nel caso che possa essere eseguita sui dati locali o in caso contrario la invia al gestore di un altro nodo contenente i dati interessati dall'interrogazione stessa, eseguendola in remoto. In entrambi i casi il DBMS locale riceve l'operazione e la esegue a tutti gli effetti andando ad interagire con la memoria di massa del nodo nella quale sono salvati i dati. La memoria di massa di ciascun nodo è limitata e può contenere una certa quantità di dati: i gestori tuttavia non sono vincolati al nodo in cui si trovano, ma possono comunicare direttamente con gli altri gestori presenti in rete e dunque accedere indirettamente alla memoria di massa degli altri nodi connessi a PariPari, utilizzando i dati contenuti o lo spazio di memorizzazione disponibile per il salvataggio di nuovi dati. Ogni nodo è in grado di salvare dati nel supporto fisico di qualsiasi altro nodo della rete, inviando al gestore remoto le operazioni da far eseguire al livello logico sottostante. Per memorizzare i dati o reperirli dalla rete, i nodi sfruttano la tabella hash distribuita messa a disposizione da PariPari: le basi di dati memorizzate nei calcolatori e lo spazio di memorizzazione messo loro a disposizione sono considerati dal

sistema come risorse condivisibili e dunque pubblicizzabili nella tabella così che, interrogando il modulo DHT, i gestori possono sapere dove trovare le risorse di cui necessitano.
La memoria messa a disposizione del sistema corrisponde all'insieme dei frammenti di memoria posseduti dai vari nodi connessi alla rete, fisicamente separati ma virtualmente interconnessi. Questo tipo di architettura logica nasconde agli utenti e alle applicazioni di basi di dati la natura distribuita del sistema, mostrando in realtà la presenza di un solo DBMS che dispone di un enorme spazio di memorizzazione. Dall'esterno esse non sono in grado di sapere se le operazioni richieste vengono eseguite localmente oppure in remoto: in effetti la distribuzione dei dati nella rete effettuata dai vari gestori non consente agli utenti o alle applicazioni di conoscere quali spazi di memoria contengono le risorse di cui necessitano.

2.6 Struttura di Pari DBMS

In questo paragrafo viene presentata la struttura interna di Pari DBMS dal punto di vista degli oggetti e dei processi che la compongono, indicandone i reciproci legami. In particolare viene descritto inizialmente il database di supporto, dopo di ché si passa alla descrizione del gestore del nodo e della struttura che ne sta alla base.

2.6.1 Database di supporto

Ogni nodo contiene al proprio interno una base di dati speciale, invisibile agli utenti e alle applicazioni, denominata *database di supporto*, che consente la memorizzazione di quei dati necessari al funzionamento del plug-in. Nella prima versione del modulo, il database di supporto è composto da quattro tabelle: la tabella DATABASES, la tabella TABLES, la tabella FIELDS e infine la tabella LOG. Le prime tre tabelle permettono di memorizzare la struttura dei database contenuti nel nodo.
La tabella DATABASES contiene al proprio interno le tuple con i riferimenti alle basi di dati locali: gli attributi di cui è composta sono NAME e HASH. NAME è una stringa che rappresenta il nome del database, mentre HASH è una stringa che contiene un codice formato dalla concatenazione del timestamp locale recuperato al momento della creazione del database con una stringa di 64 caratteri scelti casualmente tra un insieme formato dall'alfabeto minuscolo, maiuscolo e dalle dieci cifre numeriche. HASH ha la funzione di distinguere all'interno della rete quei database omonimi ma creati da utenti diversi. La chiave primaria

corrisponde solamente a NAME in quanto nello stesso nodo non possono essere presenti due database con lo stesso nome.
La tabella TABLES contiene le tuple che rappresentano le tabelle dei vari database locali: gli attributi che la compongono sono DB e NAME. DB è una stringa che indica il nome del database a cui appartiene la tabella. NAME è una stringa che rappresenta il nome della tabella. La chiave primaria corrisponde a entrambi gli attributi in quanto vi possono essere tabelle con lo stesso nome in due database diversi, ma non nello stesso database.
La tabella FIELDS contiene le tuple che rappresentano i campi delle varie tabelle: gli attributi che la compongono sono DB, TABLE, NAME, TYPE, ISNULL, PK, REF, UNIQUEID. Per ordine, DB è una stringa che rappresenta il nome del database a cui appartiene il campo in questione, TABLE è una stringa che rappresenta il nome della tabella a cui appartiene il campo, NAME è una stringa che rappresenta il nome del campo, TYPE è una stringa che rappresenta il tipo SQL del campo in questione, ISNULL è un booleano che indica se il campo accetta valori NULL, PK è un booleano che indica se il campo appartiene alla chiave primaria della tabella, REF è una stringa che contiene il nome della tabella a cui il campo si riferisce nel caso esso faccia parte di una chiave esterna, UNIQUEID è un intero che indica se il campo fa parte di un chiave unique (chiavi unique diverse vengono numerate con valori interi diversi a partire dal valore uno). La chiave primaria è formata dai primi tre attributi, in quanto una tabella di un particolare database non può contenere due attributi con lo stesso nome.

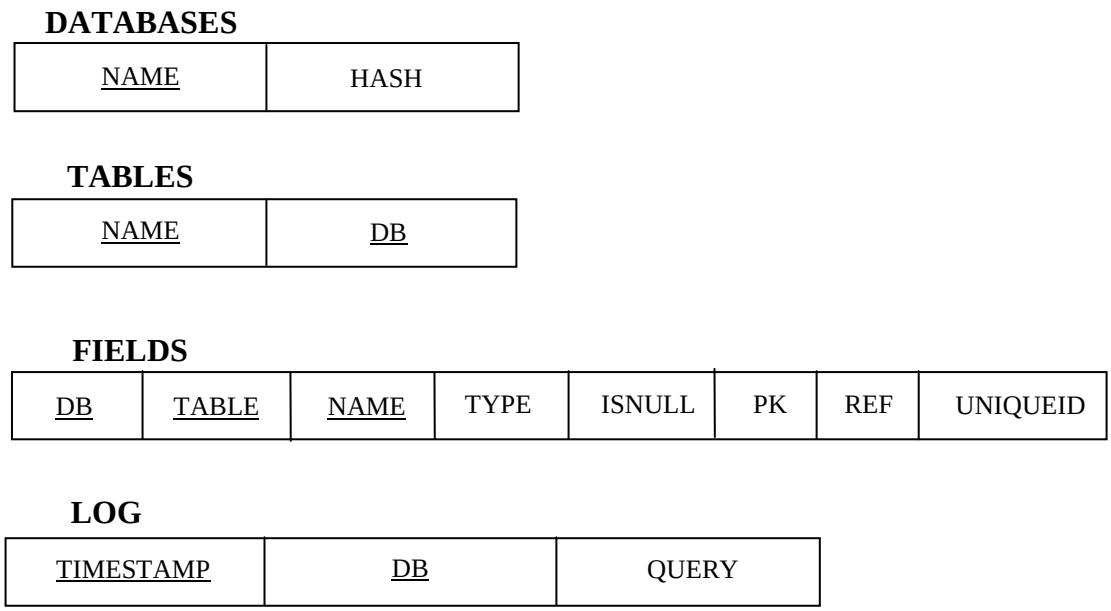


Figura 2.2: Diagramma di schema del database di supporto nella prima versione del plug-in

La tabella LOG contiene il log delle transazioni, corrispondente all'insieme delle operazioni di modifica eseguite nei vari database locali. Gli attributi di cui è composta sono TIMESTAMP, DB e QUERY: TIMESTAMP è un valore intero che rappresenta il valore dell'indicatore temporale associato all'operazione quando viene creata dal processo di sincronizzazione, DB è una stringa che rappresenta il nome del database su cui è stata effettuata l'operazione di modifica in questione, QUERY è una stringa che rappresenta la query SQL eseguita. La chiave primaria è formata dagli attributi TIMESTAMP e DB in quanto non si possono eseguire sulla stessa base di dati due operazioni con uguale timestamp.
I vari processi discussi nei paragrafi precedenti si basano sull'utilizzo delle quattro tabelle contenute nel database di supporto: i processi di sincronizzazione e re inizializzazione ricavano le informazioni fondamentali per il loro funzionamento dalla tabella LOG, mentre il processo di replicazione ricava dalle tabelle DATABASES, TABLES e FIELDS le informazioni sullo schema logico-relazionale dei database da ricreare.
Nel capitolo riguardante l'accesso remoto e il login verranno descritte le modifiche apportate al database di supporto e le motivazioni che hanno portato a tali cambiamenti.

2.6.2 Il gestore del nodo

Precedentemente è stato descritto il funzionamento del gestore del nodo, esponendone i compiti e i legami con lo strato logico superiore, dove sono situati gli utenti e le applicazioni di basi di dati, e con lo strato logico inferiore a cui appartiene il sistema di gestione delle basi di dati. Nei paragrafi successivi si procede con l'analisi della struttura interna del gestore, mettendo in luce quei processi e oggetti che lo compongono e che garantiscono il suo corretto funzionamento e allo stesso tempo del plug-in. Il gestore finora è stato visto come un unico elemento in grado di consentire la corretta esecuzione delle operazioni che gli giungono dallo strato superiore. Al di sotto di questo aspetto si nascondono in realtà tre blocchi principali dotati di funzionalità diverse e in grado di comunicare tra loro: i blocchi si dividono i compiti fondamentali di questo strato e nel loro insieme permettono il corretto funzionamento del sistema. I tre blocchi interni al gestore del nodo sono il blocco di gestione dei database locali, il blocco di gestione dei nodi e della replicazione e infine il blocco di gestione dell'aggiornamento.

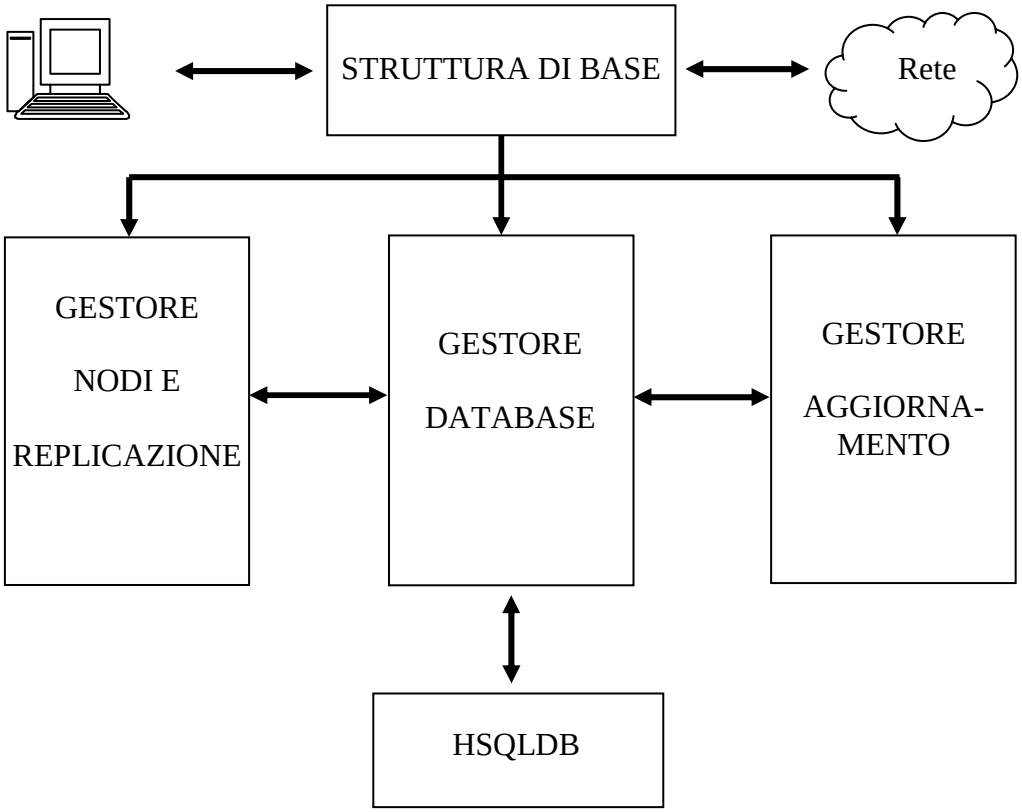


Figura 2.3: La struttura del plug-in nella sua prima versione

Il blocco di *gestione dei database locali* si occupa di garantire il normale funzionamento delle basi di dati contenute localmente, permettendone la creazione, la sincronizzazione delle operazione di modifica e la loro esecuzione nel corretto ordine in ogni replicazione.
Il blocco di *gestione dei nodi e della replicazione* si occupa della pubblicizzazione delle risorse locali, della ricerca e dell'organizzazione dei riferimenti alle basi di dati remote e dell'esecuzione del processo di replicazione per quelle basi locali che ne necessitano.
Il blocco di *gestione dell'aggiornamento* si occupa di garantire l'esecuzione del processo di aggiornamento in modo da riallineare le basi di dati una volta che il nodo si riconnette dopo un periodo di assenza dalla rete.
Ogni blocco a sua volta è costituito internamente da una serie di oggetti, alcuni dei quali sono thread, che interagiscono tra di loro o con oggetti contenuti in altri blocchi. Un thread è un processo virtuale che esegue una porzione di codice su determinati dati e che quindi rappresenta un flusso di esecuzione che svolge un compito ben definito all'interno del blocco.
Come detto in precedenza questi tre blocchi comunicano tra loro anche se svolgono compiti del tutto differenti e inoltre sono tutti supportati da un ulteriore blocco, definito struttura di base, che ne fornisce alcune funzioni elementari.

2.6.3 La struttura di base

I compiti svolti da questa sezione risultano essenziali per il corretto funzionamento del modulo anche se di per sé elementari. La struttura di base è costituita anch'essa da oggetti e da thread che consentono alcuni servizi, tra i quali la ricezione e l'invio dei messaggi tramite la rete, l'esecuzione dei comandi inviati dall'utente, lo smistamento dei messaggi ai blocchi interni, l'accesso controllato al database di supporto, la gestione delle procedure di inizializzazione e terminazione del plug-in.
Ora vengono descritti i thread e gli oggetti che compongono la struttura di base, in parte illustrati nella figura 2.4. Non viene esaminato il codice java per tali classi. L'elenco seguente comincia con la descrizione dei vari thread e delle loro funzioni per poi passare alla descrizione degli oggetti.

Command Interpreter
Thread che si occupa di interpretare i comandi inviati dall'utente attraverso l'interfaccia di PariPari, identificati dal carattere iniziale "\".

DBMS Plugin First Run
Thread che viene lanciato da DBMS ad ogni avvio del modulo. Quando viene lanciato per la prima volta in assoluto, crea il database di supporto sfruttando HSQLDB.

Message Delivery
Thread che recupera i messaggi contenuti nella coda di messaggi globale e li smista a quattro thread che li interpretano. Quelli provenienti da console vengono inviati a Command Interpreter nel caso cominciano con il carattere "\", altrimenti si tratta di operazioni SQL che vengono inviate a SQL Interpreter. A quest'ultimo vengono consegnati anche i messaggi provenienti dalla rete che cominciano con il prefisso "sql". Tutti quei messaggi che provengono dalla rete, se cominciano con il prefisso "nmg" vengono spediti a NMG Interpreter, altrimenti se cominciano con il prefisso "upd" il destinatario è Update Manager.

Network Message Receiver
Thread che riceve i messaggi provenienti dalla rete attraverso una server socket e li inserisce nella Global Message Queue.

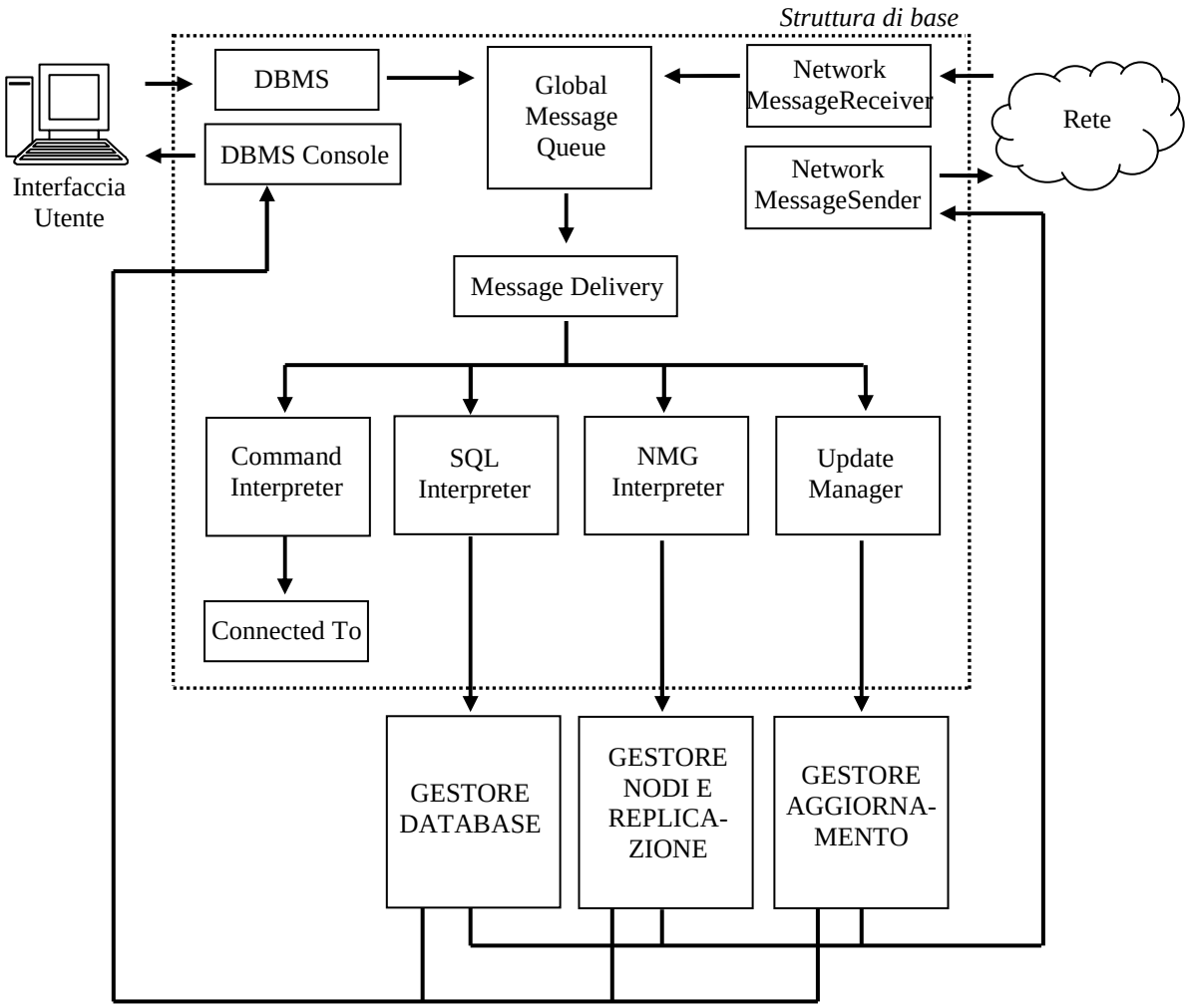


Figura 2.4: Flusso di messaggi tra struttura di base e gestori nella prima versione di Pari DBMS

Network Message Sender

Thread che invia i messaggi agli altri nodi di PariPari attraverso la rete. Ogni qual volta un oggetto deve inviare un messaggio ad un nodo, richiama Network Message Sender passandogli come parametri la stringa da inviare e l'indirizzo IP del nodo destinazione.

NMG Interpreter

Thread che fa parte del blocco di gestione dei nodi e della replicazione. Riceve da Message Delivery i messaggi diretti a tale blocco e poi li smista al suo interno. È il punto di contatto tra la struttura di base e il blocco di gestione dei nodi e della replicazione.

SQL Interpreter

Thread che riceve da Message Delivery i comandi SQL, li interpreta e li esegue nel modo più appropriato passandoli al blocco di gestione delle basi di dati locali.

Update Manager

Thread che riceve i messaggi da Message Delivery e li smista all'interno del blocco di gestione dell'aggiornamento. È il punto di contatto tra tale blocco e la struttura di base.

Connected To

È l'oggetto che rappresenta la connessione a una base di dati contenuta internamente. Quando da console arriva il comando "\use" seguito dal nome del database da utilizzare, tale oggetto imposta la propria variabile interna al nome del database locale di cui è stato richiesto l'uso. Inoltre permette di modificare lo stato della connessione stabilita.

DBMS

È un oggetto che rappresenta il plug-in stesso si occupa di inizializzare l'intera struttura e di far partire i thread che la compongono. Inoltre è il punto di contatto tra l'utente e il plug-in stesso in quanto si occupa della ricezione dei messaggi provenienti dall'interfaccia utente. Questo oggetto viene creato dal Core di PariPari quando gli giunge la richiesta di avvio del modulo.

DBMS Console

È la classe statica che gestisce l'output verso la console di PariPari. Un oggetto del plug-in che vuole stampare a video una qualunque stringa, la invia a DBMS Console indicandone il livello di verbosity: se tale livello è minore o uguale a quello impostato dall'utente allora la visualizzazione avviene, altrimenti non succede nulla.

Global Message Queue

Struttura dati, organizzata attraverso una coda FIFO, nella quale vengono inseriti i messaggi che arrivano dalla console o dalla rete. Gli inserimenti vengono effettuati da DBMS, per i messaggi che provengono dalla console, e da Network Message Receiver per quelli che giungono dalla rete. L'estrazione e lo smistamento alle varie parti del plug-in è compito esclusivo di Message Delivery.

Support DB

È l'oggetto che consente a tutti gli altri di accedere in maniera controllata e sincronizzata al database di supporto, sia in lettura che in scrittura.

Timestamp

È la classe statica che gestisce la sincronizzazione del timestamp all'interno del nodo.

2.6.4 Il gestore dei database locali

Il gestore delle basi di dati locali garantisce la corretta esecuzione delle operazioni sui database locali del nodo e nelle loro copie situate negli altri nodi della rete attraverso il processo di sincronizzazione. Come è stato fatto in precedenza con la struttura di base, nel seguito vengono descritti gli oggetti che compongono questo blocco e le loro connessioni logiche, osservabili anche nella figura 2.5.

DBSet

È una struttura dati, realizzata attraverso una lista concatenata, contenente i riferimenti a oggetti che rappresentano i database locali, identificati ognuno da un oggetto della classe Database. Tutte le classi del modulo che hanno bisogno di interagire con un database locale, comunicano con questa struttura dati attraverso il metodo *getDB* che riceve come parametro una stringa contenente il suo nome e ne restituisce il riferimento. Questa classe inoltre consente di aggiungere e togliere oggetti Database nel caso ne vengano creati di nuovi localmente o ne vengano eliminati.

Database

È l'oggetto che rappresenta un database locale. Ad ogni base di dati viene associato il nome, lo stato, una coda di transazioni e i due thread utilizzati per la sincronizzazione ed esecuzione delle operazioni. Lo stato può assumere in ogni istante uno dei tre possibili valori READY, UPDATING, PAUSED: lo stato READY identifica una base di dati allineata che può svolgere il suo normale funzionamento eseguendo tutte le operazioni che le vengono richieste; lo stato UPDATING identifica una base di dati non allineata e nella quale è in corso il processo di aggiornamento, che non permette l'esecuzione delle operazioni ma solamente il loro inserimento nella coda di transazioni in modo da eseguirle al termine dell'aggiornamento; lo stato PAUSED identifica una base di dati in cui è in corso il processo di replicazione e anche in

questo caso, come nel precedente, l'esecuzione delle operazioni viene bloccata fino a ché la base non ritorna allo stato READY, ossia quando termina la replicazione. Quando un oggetto Database viene istanziato, il suo costruttore si occupa di creare una coda di transazioni attraverso l'oggetto Transaction Queue e di lanciare il thread Transaction Synchronizer per la sincronizzazione delle operazioni di aggiornamento e il thread Transaction Executor per l'esecuzione di tutte le operazioni che lo riguardano.

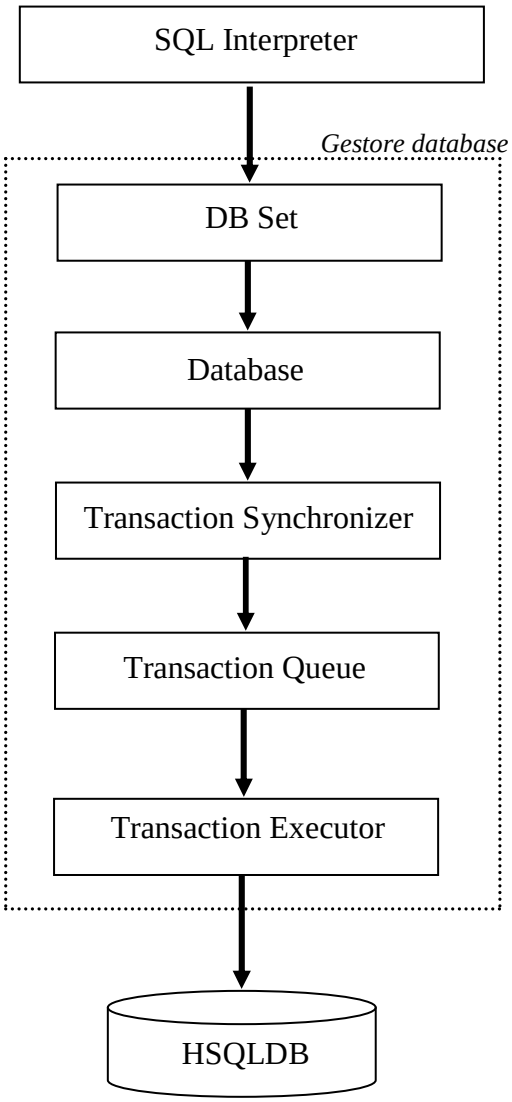


Figura 2.5: Struttura del gestore dei database locali

Transaction

È la classe che rappresenta le operazioni da eseguire sulle basi di dati e contenute nelle loro Transaction Queue. Ogni oggetto Transaction contiene al proprio interno una serie di attributi come il nome del database su cui verrà eseguita l'operazione, il timestamp associato a tale

transazione, la query SQL, lo stato dell'operazione e un valore booleano che indica se l'operazione è di aggiornamento. Nel caso quest'ultimo valore è posto a *true* allora si aggiunge un altro attributo ai precedenti: un vettore di oggetti Ack, classe interna a Transaction, che rappresentano le conferme delle replicazioni a eseguire tale operazione localmente. Infine a tutti questi si aggiunge un altro eventuale attributo definito *sender*, che rappresenta in forma di stringa l'indirizzo IP del nodo che ha richiesto l'esecuzione della query da remoto. Nel caso che l'operazione sia stata generata dal nodo locale, il sender non assume nessun valore valido. All'interno di Transaction inoltre è presente il thread TransactionSyncThread che fa parte del processo di sincronizzazione e si occupa principalmente di inviare alle repliche della base di dati in cui deve essere effettuata la transazione di aggiornamento una richiesta di esecuzione di tale operazione e a seconda delle risposte ottenute ne imposta lo stato in modo da confermarne o negarne l'esecuzione in remoto.

Transaction Queue

È la struttura dati che memorizza le operazioni da eseguire nella base di dati associata ed è organizzata secondo una coda prioritaria in modo che l'elemento estratto di volta in volta sia quello con timestamp minore. Le operazioni vengono inserite da Transaction Synchronizer ed estratte per essere eseguite da Transaction Executor.

Transaction Synchronizer

Thread che si occupa della sincronizzazione delle operazioni che riceve direttamente dal DBSet. Questo processo oltre a inserire le transazioni nella Transaction Queue interessata, interpreta i messaggi provenienti dalla rete e che iniziano con il prefisso "syn", tra i quali sono presenti messaggi di richiesta di esecuzione locale delle operazioni di aggiornamento e messaggi di conferma o meno dell'esecuzione delle operazioni. I messaggi "syn" ricevuti provengono dal thread TransactionSyncThread che interagisce con Transaction Synchronizer per garantire la sincronizzazione delle operazioni.

Transaction Executor

Thread che esegue le operazioni contenute nella Transaction Queue associata che hanno terminato correttamente il processo di sincronizzazione e che sono consentite dallo stato in cui si trova la base di dati.

2.6.5 Il gestore dei nodi e della replicazione

Il gestore dei nodi e della replicazione si occupa della pubblicizzazione delle risorse locali, della ricerca e dell'organizzazione dei riferimenti alle basi di dati remote e dell'esecuzione del processo di replicazione per quelle basi locali che ne necessitano. I processi e gli oggetti che costituiscono questo blocco si basano principalmente sull'uso di una particolare struttura dati che contiene i riferimenti ai nodi della rete che hanno risorse in comune con quelle possedute dal nodo locale. Come fatto in precedenza si descrivono brevemente gli oggetti di questa sezione e le loro connessioni logiche.

NodeSet

È la struttura dati accennata in precedenza. Contiene al proprio interno più liste di riferimenti ad oggetti: al suo interno è contenuta una lista di riferimenti a oggetti Node che rappresentano tutti i nodi che possiedono uno o più database in comune con il nodo locale e in più sono contenute tante liste, una per ciascun database locale, che contengono al loro interno gli indirizzi IP di quei nodi che hanno in comune lo stesso database. La NodeSet viene gestita e aggiornata dai tre thread Node Publicizer, Node Finder e Node Ping.

Node

È la classe che rappresenta un nodo e i suoi database. Al suo interno contiene una lista di oggetti Local Database, classe interna a Node, ognuno dei quali corrisponde a una base di dati locale. I metodi contenuti nella classe Node mantengono tale lista aggiornata nel caso di creazione locale di nuovi database o di una loro eliminazione e inoltre forniscono alcune informazioni sul nodo stesso, come il proprio indirizzo IP, e sui database locali, permettendo di conoscere o di modificarne lo stato.

Node Publicizer

Thread che si occupa della publicizzazione delle risorse locali, siano esse basi di dati contenute localmente o spazio di memoria disponibile ad ospitarne di nuove. La pubblicizzazione avviene utilizzando la tabella hash distribuita, servizio offerto dal modulo DHT. Al proprio interno il processo lancia il thread Publicizer per ogni base di dati locale che ne permette la publicizzazione. Node Publicizer si aggiorna ciclicamente in modo da pubblicizzare le risorse attuali.

Node Finder

Thread che si occupa di ricercare tutti quei nodi che possiedono database in comune con il nodo locale, aggiornando di volta in volta le liste della NodeSet. La ricerca avviene tramite il thread interno Finder che viene lanciato per ogni base di dati contenuta localmente e che effettua la ricerca dei nodi tramite DHT. Node Finder inoltre invia a tutti i nodi recuperati un messaggio che li invita ad aggiungere nelle proprie NodeSet le informazioni riguardanti tale nodo e i suoi database contenuti.

Node Ping

Thread che si occupa di mantenere aggiornata la NodeSet. Tale processo invia un messaggio a ogni nodo contenuto nella NodeSet per verificarne la presenza, associandone il timestamp attuale, dopo di ché si mette in attesa delle risposte per un certo intervallo di tempo, alla scadenza del quale ripete la procedura. Per tutti quei nodi che hanno risposto entro la scadenza del timeout, il loro timestamp viene aggiornato al nuovo valore attuale. Node Finder come Node Publicizer si aggiorna ciclicamente.

Node Controller

Thread che assieme a Node Ping mantiene aggiornata e controllata la NodeSet. Per prima cosa esegue un confronto tra il valore del timestamp associato a ogni nodo con quello attuale e nel caso la loro differenza sia maggiore o uguale a un determinato valore temporale allora il nodo viene considerato inattivo e ogni suo riferimento viene eliminato dalle liste della NodeSet. Al termine dell'aggiornamento della NodeSet, il thread controlla esaminando le liste di indirizzi IP la quantità di oggetti contenuti in modo che se una lista contiene al proprio interno un numero di indirizzi IP inferiore a una certa soglia allora per il database associato a tale lista viene lanciato il processo di replicazione. Node Controller si aggiorna ciclicamente.

Replication Client

Thread che si occupa del processo di replicazione dal lato client. Viene lanciato da Node Controller quando si rileva il rischio di indisponibilità per un database locale. Il thread inizialmente decide se il database locale verrà utilizzato per la replicazione a caldo o per rispondere alle query. Nel primo caso Replication Client ricerca tramite DHT un nodo da contattare come server per il processo di replicazione e una volta ottenuto procede inviandogli i messaggi di replicazione in modo da ricreare il database nel server.

Replication Set Client

È la struttura dati, organizzata secondo una lista concatenata, contenente gli oggetti Replication Client attivi localmente. Inoltra i messaggi ricevuti da NMG Interpreter al corretto Replication Client locale.

Replication Server

Thread che si occupa del processo di replicazione dal lato server. Viene creato direttamente da NMG Interpreter nel momento in cui al nodo giunge una richiesta di replicazione che può essere accettata. Replication Server riceve i messaggi contenenti le query provenienti dal client corrispondente, ricevuti attraverso il Replication Set Server, e si occupa della loro esecuzione in modo da creare la copia del database richiesto.

Replication Set Server

È la struttura dati contenente gli oggetti Replication Server attivi localmente. Inoltra i messaggi ricevuti da NMG Interpreter al corretto Replication Server. Ad essi sono destinati i messaggi di replicazione che contengono le operazioni SQL da eseguire.

2.6.6 Il gestore dell'aggiornamento

Il gestore dell'aggiornamento si occupa di garantire l'esecuzione del processo di aggiornamento in modo da riallineare le basi di dati una volta che il nodo si riconnette dopo un periodo di assenza dalla rete. Come fatto in precedenza si descrivono brevemente gli oggetti di questa sezione e le loro connessioni logiche.

Update Manager

Thread che si occupa di gestire il processo di aggiornamento: crea e lancia tanti thread DBUpdate Client quante sono le basi di dati locali da aggiornare. Dispone di una coda interna in cui vengono depositati i messaggi scambiati tra DBUpdate Client e Server. Inoltre contiene un metodo che permette di ordinare l'aggiornamento di una replica appena effettuata prima di dichiararla disponibile alla rete.

DBUpdate Client

Thread che invia le richieste di aggiornamento ed elabora gli aggiornamenti ricevuti eseguendo le transazioni contenute. Esiste un thread per ogni base di dati contenuta localmente e dispone di una coda interna dove vengono memorizzati i messaggi provenienti da Update Manager.

DBUpdate Server

Thread in esecuzione su ciascun nodo che deve inviare gli aggiornamenti ai client. Effettua il recupero degli aggiornamenti da far eseguire al client per potersi riallineare. Dispone di una coda interna nella quale Update Manager deposita i messaggi provenienti dai vari client.

3. Integrazioni di Pari DBMS

In questo capitolo verranno descritte le modifiche apportate all'interno del plug-in Pari DBMS, in particolare la sua integrazione con il modulo Diesel e con il plug-in Connectivity. Per ognuna di esse verranno precedentemente analizzati i motivi che hanno condotto all'introduzione di questi cambiamenti nel sistema. Verrà dato maggior spazio alla descrizione della parte di integrazione con Connectivity in quanto tale lavoro è stato svolto da me, mentre la parte di integrazione con Diesel verrà solamente accennata senza entrare nei minimi dettagli in quanto realizzata dal collega Deniz Tartaro Dizmen. Si raccomanda la lettura della sua tesi per ottenerne informazioni più approfondite.

3.1 Integrazione con Diesel

Nella prima versione del plug-in ogni nodo possiede una lista interna, la Node Set, che contiene i riferimenti agli altri nodi della rete che hanno database in comune con esso. Ogni nodo mantiene aggiornata la Node Set inviando a tutti i nodi in essa contenuti un messaggio che ne indica l'attuale presenza nella rete (messaggio di ping) e mettendosi successivamente in attesa delle risposte per un certo intervallo di tempo. Nel momento in cui il nodo si risveglia, controlla quali nodi della Node Set hanno inviato precedentemente il messaggio di risposta e per tutti quei nodi per cui non è arrivato ne elimina il riferimento dalla lista. Questo processo che permette a un nodo di individuare quali altri nodi, che offrono lo stesso servizio, risultano attivi in un particolare istante comporta l'invio nella rete di un numero di messaggi pari a quello dei nodi contenuti nella Node Set locale. Nella rete dunque circola una grande quantità di messaggi destinati solamente alla verifica della disponibilità dei nodi che fanno parte del sistema Pari DBMS. L'integrazione del plug-in con il modulo Diesel ha come obiettivo principale quello di ridurre la quantità di messaggi che i nodi si scambiano tra loro. Diesel infatti gode di un algoritmo che consente a tutti quei nodi che offrono un particolare servizio di inviarsi messaggi di ping in maniera intelligente e distribuita all'interno della rete, evitando di sovraccaricarla con tali messaggi. Il plug-in dunque in questa nuova versione affida a tale modulo la gestione efficiente dei messaggi di ping tra i nodi di Pari DBMS, che precedentemente veniva effettuata dal thread Node Ping. Inoltre garantisce un meccanismo che permette a tutti i nodi che godono di un particolare servizio di venire a conoscenza della disconnessione dalla rete di uno di loro: nel momento della disconnessione Diesel è in grado di avvertire il resto dei nodi inviando un messaggio di norifica. Tutti i nodi nel momento di ricezione del messaggio andranno a

verificare di contenere nella Node Set locale il riferimento al nodo appena disconnesso e nel caso lo contengano andranno ad aggiornare la lista eliminando il riferimento stesso. In questo modo un nodo non dovrà più attendere per un certo intervallo di tempo l'arrivo delle conferme ai messaggi di ping inviati e valutare successivamente al termine dell'attesa quali nodi sono ancora presenti nella rete e quali non lo sono più.

3.2 Integrazione con Connectivity

Come abbiamo accennato nell'introduzione del capitolo precedente, il plug-in nella sua prima versione utilizzava in ogni nodo per la comunicazione con la rete una server socket con cui riceveva i messaggi dalla rete e una serie di socket con cui inviare i messaggi nella rete. I processi che permettevano e che tuttora permettono la comunicazione tra i nodi di Pari DBMS e la rete corrispondono ai due thread Network Message Sender, per l'invio dei messaggi, e Network Message Receiver per la loro ricezione: le socket da loro utilizzate venivano ottenute direttamente dalle librerie di Java. Network Message Receiver utilizzava una server socket in attesa di nuove connessioni sulla porta 1600, mentre Network Message Sender ogni qual volta doveva inviare un messaggio a un nodo attraverso la rete utilizzava una nuova socket di comunicazione, anche nel caso avesse già comunicato in precedenza con quello stesso nodo. Questo comportava un enorme spreco di socket di comunicazione che venivano utilizzate solo per l'invio e la ricezione di un unico messaggio. In questa nuova versione il plug-in interagisce con il plug-in Connectivity per il recupero delle socket necessarie alla comunicazione e inoltre gode di un meccanismo di gestione delle socket che ne consente il riutilizzo. La nuova gestione delle socket comporta dei cambiamenti nel funzionamento dei due thread di comunicazione con la rete, rendendoli più complessi, ma in futuro, quando il plug-in DBMS verrà integrato anche il modulo Crediti, porterà a dei vantaggi in termini di riduzione del costo associato alla richiesta di nuove risorse: il riutilizzo delle socket permetterà di risparmiare crediti da utilizzare per richiedere altre risorse in seguito. Nel seguito vengono descritte le modifiche apportate ai due thread di comunicazione con la rete che garantiscono il funzionamento della nuova gestione delle socket.

3.2.1 Modifiche a Network Message Sender

In questa nuova versione di Pari DBMS, Network Message Sender (NMS) viene dotato di una lista concatenata, definita *lista delle connessioni*, contenente al proprio interno le socket di comunicazione richieste per l'invio dei messaggi nella rete. Questa lista in realtà contiene oggetti particolari, ognuno dei quali è composto dall'indirizzo IP del nodo con cui si è stabilita la connessione in precedenza, la socket utilizzata dal nodo locale per inviargli i messaggi e i flussi d'ingresso e di uscita associati a tale socket. Il processo di invio dei messaggi avviene nel seguente modo. Network Message Sender recupera dalla propria coda di messaggi il messaggio da inviare e l'indirizzo IP del nodo di destinazione. Per prima cosa il thread controlla che l'indirizzo IP sia presente in un oggetto contenuto nella lista delle connessioni locale. Nel caso ciò non avvenga allora significa che il nodo locale non ha mai inviato messaggi al nodo destinazione in precedenza: la prima volta che un nodo invia un messaggio a un altro nodo della rete, il thread NMS effettua una richiesta al modulo Connectivity in modo da ottenere una socket di comunicazione, rappresentata dall'oggetto *LimitedSocketAPI*. Connectivity mette a disposizione socket limitate nella larghezza di banda. Quando riceve la socket il thread NMS crea un nuovo oggetto da inserire nella lista delle connessioni passandogli l'indirizzo IP del nodo destinatario, la socket ottenuta per la comunicazione e i flussi associati in modo da poterli recuperare successivamente. A questo punto invia attraverso il flusso di uscita il messaggio recuperato inizialmente associandogli il tempo attuale, attraverso il metodo System.currentTimeMillis, che viene utilizzato da Network Message Receiver per ordinare i messaggi che gli arrivano. Ritornando a prima, nel caso in cui invece l'indirizzo IP del nodo destinatario sia presente in un oggetto della lista delle connessioni allora significa che precedentemente è stato inviato un messaggio a tale nodo e che si è stabilita una connessione da riutilizzare. Il thread NMS prima di riutilizzare la socket associata, verifica che i flussi siano aperti in quanto una eventuale disconnessione del nodo destinatario potrebbe averli chiusi e che nel flusso d'ingresso non sia contenuto un messaggio di tipo "End Connection", il cui significato verrà spiegato nel prossimo paragrafo. Se le verifiche vanno a buon fine allora il thread NMS può riutilizzare la socket per la comunicazione andando a inserire nel flusso d'uscita recuperato il messaggio da inviare assieme al tempo corrente. Nel caso contrario il thread NMS deve procedere eliminando la socket non più utilizzabile dalla lista delle connessioni locali, chiundendone i flussi associati, e creando una nuova connessione con il nodo destinazione, come in precedenza.

3.2.2 Modifiche a Network Message Receiver

Nella nuova versione del plug-in anche Network Message Receiver (NMR) viene dotato della lista delle connessioni come avveniva per il corrispondente Network Message Sender. Questa lista contiene i riferimenti alle socket di comunicazione utilizzate dal nodo locale per ricevere i messaggi che giungono dalla rete. In particolare possiede oggetti che memorizzano al loro interno l'indirizzo IP del nodo da cui provengono i messaggi, il socket di ricezione di tali messaggi e i flussi d'ingresso e d'uscita ad esso associati. Il processo di ricezione avviene nel seguente modo. Il thread NMR al suo avvio effettua una richiesta a Connectivity per ottenere una server socket: Connectivity restituisce una server socket limitata nella larghezza di banda, rappresentata dall'oggetto *LimitedServerSocketAPI*, che ritorna socket di comunicazione limitate nella larghezza di banda sulle connessioni che vengono accettate (oggetti LimitedSocketAPI) e che non accetta più connessioni quando la sua banda di ingresso è stata completamente assegnata. La server socket è in attesa delle nuove connessioni sulla porta 1600 per un certo intervallo di tempo. Quando viene accettata una nuova connessione sulla server socket, viene creato una socket di comunicazione per il recupero dei messaggi inviati. Il thread NMR a sua volta crea un oggetto da inserire nella lista delle connessioni, memorizzandone l'indirizzo IP del nodo che ha effettuato la nuova connessione e la socket di comunicazione assegnata alla ricezione dei messaggi che giungono da quel mittente assieme ai flussi d'ingresso e uscita associati. Nel caso che sia già presente all'interno della lista delle connessioni un oggetto riferito a tale indirizzo IP, il thread lo sostituisce con il nuovo oggetto creato. Successivamente effettua il recupero del messaggio inviato assieme al valore corrente associato. Questo valore viene utilizzato dal thread per mantenere ordinati i messaggi che sono giunti fino all'arrivo della nuova connessione nei flussi d'ingresso delle socket di comunicazione attive e presenti nella lista delle connessioni. Infatti a questo punto il thread NMR esamina la lista delle connessioni ed estrae i messaggi contenuti in tutte quelle socket che hanno i rispettivi flussi d'ingresso occupati. Per ogni messaggio viene recuperato anche il valore corrente associato al momento dell'invio e viene creato un oggetto contenente il messaggio stesso, il valore corrente associato e l'indirizzo IP del nodo mittente. Gli oggetti vengono inseriti all'interno di una coda a priorità nella quale vengono ordinati secondo valori correnti crescenti. Dopo di ché viene aggiunto anche l'oggetto creato per il messaggio giunto con la nuova connessione. Terminato di ordinare gli oggetti il thread estrae dalla coda i messaggi a partire da quello con valore corrente minore e li inserisce nella coda globale di messaggi in modo che vengano distribuiti tra i vari oggetti del sistema.

Precedentemente si è affermato che la server socket si mette in attesa di nuove connessioni sulla porta di benvenuto per un tempo limitato: la scadenza di tale timeout di attesa implica il non arrivo di nuove connessioni. In questo caso si procede come nel caso di arrivo di una nuova connessione andando a recuperare i messaggi giunti fino alla scadenza del timeout dalle socket contenute nella lista delle connessioni e con i rispettivi flussi di input occupati. I messaggi vengono poi ordinati e inseriti nella coda globale di messaggi. Al termine dell'inserimento il thread NMR ritorna al punto di partenza mettendosi in attesa sulla propria server socket dell'arrivo di nuove connessioni da elaborare.

Nella gestione delle socket nel thread NMR si deve anche considerare il fatto che la server socket ottenuta attraverso Connectivity è limitata nella sua banda di ingresso che quando viene esaurita completamente non permette di accettare le nuove connessioni in arrivo. Il thread NMR nel momento in cui accetta una nuova connessione deve valutare quanta banda di ingresso della server socket è stata assegnata alle socket di comunicazione create per accettare le connessioni precedenti. Quando questo valore si avvicina al valore massimo stabilito per la banda della server socket allora il thread deve liberare banda andando a chiudere la socket contenuta nella lista delle connessioni che non viene utilizzata da più tempo o che è stata recuperata di meno. Stabilita la socket da chiudere, prima di eliminarla si deve avvisare dall'altro lato il nodo dal quale tale socket riceve i messaggi inviandogli un messaggio di tipo "End Connection" attraverso il proprio flusso di output. Il messaggio comporta la chiusura nel nodo mittente della socket corrispondente e la sua eliminazione dalla lista delle connessioni associata al thread NMS locale.

4. Accesso remoto e Login

Nel seguente capitolo verranno trattate due problematiche importanti ma tra loro collegate: l'accesso remoto e il login. Entrambe le problematiche e i meccanismi che permettono la loro risoluzione sono stati presi in considerazione e realizzati da me e dal collega Deniz Tartaro Dizmen. Tuttavia questo elaborato non prevede una descrizione nei minimi dettagli di tutti gli aspetti riguardanti l'accesso remoto che si possono trovare nell'elaborato del mio collega, ma contiene una descrizione dettagliata riguardo alla progettazione e realizzazione del meccanismo di login. Ora verranno analizzati assieme i due aspetti nel contesto di Pari DBMS e successivamente verranno trattati singolarmente, entrando in dettaglio maggiormente per la parte di login.

Come è stato detto all'inizio del capitolo 2 un utente o un'applicazione che usufruisce del plug-in Pari DBMS deve poter effettuare operazioni sulle basi di dati di cui necessita, sia che esse siano contenute nel nodo in cui ci si trova al momento della richiesta sia che siano situate in altri nodi della rete. Nella prima versione del modulo, chiunque volesse connettersi ad una base di dati poteva farlo solamente se il nodo in cui era situato la conteneva localmente. Dunque un utente che voleva utilizzare una particolare base di dati ma che non era contenuta localmente, poteva avere due alternative: o era in grado di conoscere il nodo in cui la base era contenuta localmente e dal quale poteva accedere direttamente alla base, o in mancanza di tale informazione doveva rinunciare ai dati di cui aveva bisogno. In questa nuova versione invece, un utente è in grado di accedere ai dati di cui ha bisogno da qualunque nodo si trovi, senza dover disporre necessariamente di informazioni che riguardino i database contenuti da un nodo: ciò è permesso dal meccanismo di accesso remoto. Tale meccanismo consente di celare agli utenti e alle applicazioni di basi di dati la natura distribuita del sistema: chiunque deve utilizzare un database d'ora in avanti non dovrà essere a conoscenza del nodo in cui è situata effettivamente la base di dati da utilizzare, ma dovrà semplicemente richiedere l'uso del database necessitato in locale e poi lasciare al sistema stesso il compito di connettersi a tale database. Un utente in questo modo non sarà in grado di comprendere se le operazioni da lui richieste vengano effettivamente svolte in locale o in remoto: ciò che interessa è che ne sia garantita l'esecuzione.
Finora è stato detto che un utente può connettersi a qualunque base di dati necessiti: ciò non è del tutto corretto in quanto un utente deve accedere solamente a quelle basi di dati da lui create o alle quali può avere accesso perché considerato un utente fidato da chi l'ha creata.

Al momento della creazione di un nuovo database nella rete bisognerà verificare le credenziali dell'utente creatore. La verifica deve avvenire anche nel caso l'utente voglia utilizzare un particolare database, sia che sia contenuto in locale che in remoto. Se la verifica risulta positiva allora l'utente potrà usufruire dei dati richiesti altrimenti non potrà farlo. Il meccanismo che permette la creazione e la verifica delle credenziali per la creazione e l'utilizzo di un database si definisce meccanismo di login e verrà discusso dettagliatamente nel prossimo paragrafo.

4.1 Login

4.1.1 Modifiche al sistema

Prima di procedere con la descrizione del meccanismo di login conviene chiarire alcuni aspetti riguardanti il sistema. Nella prima versione del plug-in, si era deciso di non permettere il salvataggio di due database omonimi nello stesso nodo, anche se ciò era stato permesso nella rete grazie all'introduzione del codice hash. Due database omonimi ma dotati di codici hash diversi potevano essere creati nella rete ma non potevano essere memorizzati contemporaneamente nello stesso nodo. Le publicizzazioni e le ricerche dei database effettuate tramite DHT utilizzate dai thread tenevano conto delle regole stabilite utilizzando come chiave il nome del database e il codice hash associato, la coppia di valori che rendeva distinguibili i database tra di loro e tra i loro omonimi. In questa nuova versione sono stati effettuati dei cambiamenti stabilendo nuove regole e modificando le precedenti. Per prima cosa si è stabilito che nella rete possono essere presenti due o più database omonimi purché essi siano creati da utenti diversi tra loro. Ogni nodo può contenere al proprio interno database omonimi purché differiscano nel codice hash e negli utenti che l'hanno creato. In questo modo viene negato a un qualsiasi utente di creare due o più basi di dati assegnandone lo stesso nome: nella rete non potranno essere presenti due basi di dati con lo stesso nome, stesso utente e codici hash diversi. Le basi di dati possono essere identificate univocamente tra loro attraverso la coppia costituita dal nome del database e codice hash associato, come nella versione precedente, o dalla nuova coppia formata dal nome del database e dal nome dell'utente. Entrambe le coppie vengono utilizzate come chiavi per la ricerca e la publicizzazione delle basi di dati su DHT, anche se ad esse viene aggiunta un'altra chiave costituita solamente dal nome dell'utente il cui utilizzo verrà spiegato prossimamente. Per permettere che all'interno dello stesso nodo vi possano essere due

database omonimi si è pensato di andare a modificare alcune tabelle del database di supporto[3]: per quanto riguarda la tabella DATABASES la modifica riguarda la chiave primaria che non risulta più essere solamente l'attributo NAME ma da ora in avanti è costituita da entrambi gli attributi NAME e HASH in quanto ora ogni nodo può contenere al suo interno database con nomi uguali ma che devono differire per il codice hash; per quanto riguarda le altre tabelle TABLES, FIELDS e LOG viene aggiunto il campo HASH e viene reso parte delle rispettive chiavi primarie in modo da poter permettere al nodo di contenere due o più database con lo stesso nome ma creati da utenti diversi e per distinguere il corretto database cui sono riferite le tabelle, i campi e le transazioni.

Finora si è parlato di utenti senza definire come il meccanismo di login li distingue nel sistema: ad ogni utente vengono associate due credenziali corrispondenti al nome utente (o username) e alla password. Un utente nel momento in cui crea il suo primo database all'interno del sistema definisce il proprio nome e associandone una password di riconoscimento: tale database viene creato e salvato all'interno del database di supporto locale assieme al nome dell'utente che l'ha creato e alla password associata. Per garantire il mantenimento di questi dati e non disponendo attualmente di un plug-in che offra servizi di login, si è pensato di dotare il database di supporto di una nuova tabella definita USERS, costituita di tre attributi: DBNAME è una stringa che rappresenta il nome del database, USERNAME è una stringa che rappresenta il nome dell'utente che lo ha creato e infine PASSWORD è una stringa che contiene la password di identificazione dell'utente stesso. Questa nuova tabella viene interrogata dal meccanismo di login ogni qual volta si devono verificare le credenziali di un qualsiasi utente che intende accedere o creare un database. La chiave primaria è formata dai primi due attributi DBNAME e USERNAME, in quanto all'interno di un nodo non è possibile che lo stesso nome utente crei due o più basi di dati omonime. Inoltre all'interno del sistema non possono esistere due utenti con lo stesso nome ma password diverse, in quanto ogni utente si distingue da un altro attraverso il proprio nome e non attraverso la password. La password associata all'utente è unica e viene impostata al momento della creazione del suo primo database, anche se può accadere che due utenti diversi, che quindi possiedono nomi utenti differenti, godano della stessa password.

La nuova tabella USERS comporta a sua volta un cambiamento nel processo di replicazione, in particolare per quel che riguarda i messaggi di replicazione inviati. Prima dell'inserimento di tale tabella, il processo di replicazione inviava due tipi di messaggi[4]: quelli contenenti informazioni sullo schema logico-relazionale della base di dati che ne permettevano la corretta

[3] Vedi paragrafo 2.6.1 riguardante il database di supporto.
[4] Vedi paragrafo 2.4 sulla replicazione delle basi di dati.

replicazione e quelli contenenti i dati posseduti dalla base di dati stessa. Il processo di replicazione deve inviare oltre a questi anche un nuovo tipo di messaggi che permettano di aggiornare le tabelle USERS dei nodi in cui vengono replicate le basi di dati: questi contengono al loro interno le credenziali dell'utente che ha creato la base di dati da replicare. In questo modo le nuove copie realizzate possono essere utilizzate correttamente garantendo l'accesso solamente agli utenti autorizzati.
Il meccanismo di login deve garantire il pieno rispetto delle regole che riguardando il sistema e gli utenti esposte in questo paragrafo.

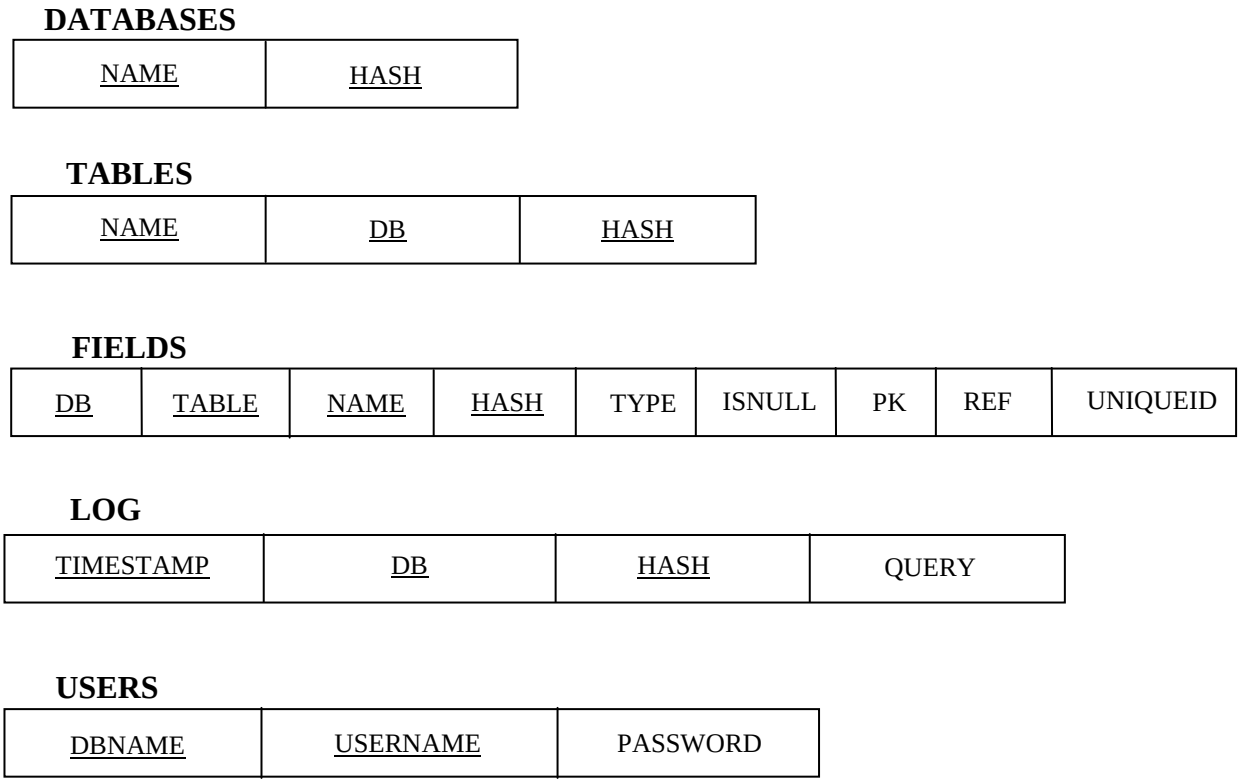


Figura 4.1: Diagramma di schema del database di supporto nella nuova versione del plug-in

4.1.2 Procedura di login

Uno degli obiettivi principali di qualunque DBMS è garantire la privatezza dei dati: ciascun utente, identificabile univocamente mediante un nome all'atto di interagire con il DBMS, può svolgere determinate azioni sui dati. La procedura di login fornisce al sistema i meccanismi di autorizzazione e controllo di qualunque utente che intende usufruire delle basi di dati messe a sua disposizione. Nel sistema distribuito Pari DBMS essa permette principalmente di impostare e verificare le credenziali associate a un utente che ne interagisce per la creazione e l'utilizzo

delle basi di dati contenute. Si attiva in due diverse circostanze: al momento della creazione di un nuovo database e prima di accedere a una qualsiasi base di dati.
Prima di tutto occorre ricordare che si è stabilito che la creazione di un database può avvenire solo localmente e non da remoto e nel caso che il nodo locale possieda disponibilità a ospitare nuove basi di dati.
La seguente descrizione fa riferimento all'immagine presente in *Appendice A*, dove è disegnato uno schema a blocchi che rappresenta la procedura di login all'interno del processo di creazione di un database.
Un utente qualsiasi, dotato di username e password proprie, nel momento in cui vuole creare un database all'interno del sistema viene sottoposto a una serie di controlli in modo da garantire il rispetto delle regole definite nel paragrafo precedente. L'utente nel messaggio di richiesta della creazione del database inserisce il nome del database a cui fa seguire il proprio username e la password associata, scrivendo un messaggio del tipo "CREATE DATABASE DBNAME: USERNAME: PASSWORD". Il messaggio di creazione viene ricevuto dal sistema che attraverso Message Delivery lo smista all'interprete delle operazioni SQL, il thread SQL Interpreter, che a sua volta lo inserisce nella coda di messaggi del thread DBCreator che si occupa del processo di creazione del database. DBCreator recupera dal messaggio ricevuto le informazioni riguardanti il nome del database da creare, il nome dell'utente che lo vuole creare e la password associata, dopo di ché effettua una ricerca su DHT usando come chiave di ricerca il solo nome dell'utente. Questa ricerca consente di recuperare gli indirizzi IP di tutti quei nodi che contengono al loro interno database creati precedentemente da tale utente e dunque permette di verificare se quell'utente ha già interagito con il sistema andando a crearvi basi di dati in passato. Due sono i possibili risultati che si possono ottenere: possono venire restituiti una serie di indirizzi IP oppure può non essere restituito nessun indirizzo IP. Il verificarsi del secondo caso comporta che all'interno della rete non vi sia presenza di alcun database creato da tale utente. L'utente dunque non ha mai interagito con il sistema precedentemente per il quale risulta a tutti gli effetti un nuovo utente: la creazione del database può essere effettuata correttamente attivando la procedura di login per il salvataggio delle informazioni che riguardano il database stesso e l'utente che lo vuole creare.
Tale procedura si affida alla classe *Authorization* in grado di interagire con il database di supporto locale, in particolare con la tabella USERS in cui sono contenute per ogni database locale le informazioni riguardanti l'utente che l'ha creato. Authorization è dotata al proprio interno di due metodi: il metodo *setCredentials* che consente di impostare le credenziali a un database nel momento della sua creazione e riceve come parametri il nome del database creato,

il nome dell'utente che l'ha creato e la password associata; il metodo *checkUsers* che consente di verificare il nome utente e la password associati al database locale che si vuole utilizzare, ricevendo come parametri il nome del database di cui bisogna verificare le credenziali associate, il nome e la password dell'utente che intende utilizzarlo e restituendo un valore booleano che indica la correttezza o meno della verifica. Il metodo setCredentials effettua un'operazione di inserimento all'interno della tabella USERS dei valori passati, mentre il metodo checkUsers effettua un'operazione di selezione sulla tabella USERS dei valori passati in modo da controllarne la corrispondenza con quelli salvati al proprio interno.

Ritornando al discorso precedente, nel momento della creazione del database la procedura di login prosegue richiamando il metodo setCredentials della classe Authorization che inserisce all'interno della tabella USERS del database di supporto la tupla contenente come valori il nome del database creato, il nome dell'utente che lo ha creato e la sua password associata. D'ora in avanti all'interno del sistema tale database viene identificato dall'utente che l'ha creato e chiunque volesse utilizzarlo dovrà disporre del nome utente e della password associate dal creatore stesso. Inoltre da questo momento il sistema interagisce con un nuovo utente identificato al suo interno unicamente dal proprio username e dalla password associata.

Nel caso invece che la ricerca precedente abbia fornito una lista di indirizzi IP, significa che l'utente che ha richiesto la creazione della base di dati ha già interagito in passato con il sistema creando al suo interno dei database. A questo punto interviene la procedura di login per effettuare la verifica delle credenziali che appartengono a tale utente, in particolare si vuole verificare se la password assegnata nel messaggio di creazione corrisponde alla password associata a tale utente, stabilita alla creazione del suo primo database nel sistema. In questo caso viene sfruttato il metodo checkUsers della classe Authorization che effettua il confronto tra nomi utente e le password associate. In particolare viene recuperato l'indirizzo IP di uno dei nodi restituiti dalla ricerca e ad esso viene inviato un messaggio contenente lo username e la password associate alla creazione del database. Il nodo contattato contiene almeno un database creato da un utente con lo stesso username di quello passato e quando riceve il messaggio precedente esegue il metodo checkUsers di Authorization con cui confronta la password ricevuta con quella salvata nella tabella USERS del database di supporto locale e associata all'utente omonimo. Se le due password corrispondono allora le credenziali utilizzate per la creazione sono corrette e l'utente viene riconosciuto dal sistema, altrimenti le credenziali non sono corrette e l'utente non viene riconosciuto dal sistema e non potrà proseguire con la creazione del database. Nel primo caso il sistema deve verificare che l'utente appena riconosciuto valido non abbia precedentemente creato un database avente lo stesso nome di

quello che vuole creare. Viene dunque effettuata una nuova ricerca su DHT con chiave di ricerca la coppia formata dal nome del database da creare e dallo username del creatore in modo da controllare se nella rete sia già presente tale database creato da tale utente. Come prima, si presentano due possibilità: nel caso la ricerca restituisce una serie di nodi allora ciò significa che tale utente ha già creato in precedenza tale database e non potrà crearlo nuovamente, mentre nel caso che la ricerca non restituisca alcun nodo allora l'utente deve ancora creare tale database e potrà farlo successivamente. La creazione del database, come descritto precedentemente, comporta il salvataggio delle credenziali associate al database creato all'interno della tabella USERS del database di supporto locale, utilizzando il metodo setCredentials di Authorization.
Questi controlli garantiscono che all'interno del sistema due utenti diversi non possano godere dello stesso username e di password diverse e che lo stesso utente non possa creare due database con lo stesso nome. La creazione di un database è autorizzata dal sistema solo per i nuovi utenti e per quelli riconosciuti che devono ancora crearlo.

Finora si è parlato della procedura di login nel caso un utente voglia creare un database, ma non nel caso in cui un utente voglia utilizzare un particolare database. La procedura di login prevede che a un database vengano assegnate le credenziali dell'utente creatore durante la propria creazione e che solo chi è in possesso di tali credenziali possa accedervi in seguito. Prima di accedere a un database deve essere effettuata una verifica delle credenziali utilizzate: solo se il nome utente e la password corrispondono a quelle associate al database al momento della sua creazione allora l'utente è autorizzato all'accesso e all'utilizzo. Tale verifica deve essere effettuata sia che l'accesso avvenga per un database locale sia che avvenga in remoto. Anche se le due procedure di accesso differiscono tra loro, prevedono entrambe l'esecuzione della procedura di login prima di consentire a un utente l'utilizzo del database di cui necessita. Le modalità di accesso a una base di dati verranno presentate nel prossimo paragrafo, tuttavia per il momento occorre sapere che in entrambi gli accessi si arriverà ad un punto in cui dovrà essere effettuata la verifica delle credenziali associate al database da utilizzare. In quel momento sia il nodo locale che il nodo remoto in cui può essere svolta la verifica utilizzano il metodo checkUsers della classe Authorization passando come parametri il nome del database che si vuole utilizzare, il nome dell'utente che intende usufruirne e la password ad esso associata: il metodo recupera dalla tabella USERS del database di supporto locale la tupla corrispondente al database che l'utente vuole utilizzare ed effettua un confronto tra la password passata come parametro e definita al momento dell'uso del database e quella recuperata dalla tupla ossia quella associata al database dall'utente che l'ha creato. Se le due password corrispondono allora

l’utente che intende utilizzare quel database è in possesso delle credenziali dell’utente che lo ha creato e dunque è a tutti gli effetti autorizzato ad accedervi e a eseguire operazioni su di esso. In caso contrario, l’utente sta tentando di accedere al database con credenziali che non corrispondono a quelle stabilite al momento della creazione del database e dunque gli verrà negato l’accesso.

4.2 Accesso a una base di dati

All’inizio di questo capitolo si è accennato al fatto che un utente nel momento in cui usufruisce del servizio offerto da Pari DBMS deve poter accedere alle basi di dati che necessita, sia nel caso queste siano contenute nella memoria di massa del calcolatore a cui si è connessi sia che siano contenute nella memoria di altri calcolatori della rete. Il sistema dispone di un meccanismo definito *protocollo di accesso remoto* che consente di accedere alle basi di dati situate in nodi remoti. Si tratta di un meccanismo interno al sistema e dunque non visibile dall’utente esterno che non è in grado di comprendere se il database a cui si è connesso e con cui successivamente interagisce sia contenuto localmente o in remoto. Ora si descrive il processo che permette a un utente di connettersi a un particolare database, indicando gli oggetti del sistema richiesti per tale scopo e rappresentato nello schema a blocchi della figura in *Appendice B.* Per una visione più approfondita dell’accesso a una base di dati e del protocollo dell’accesso remoto si consiglia la lettura dell’elaborato del collega Deniz Tartaro Dizmen.

Un utente, dotato di username e password proprie, nel momento in cui vuole utilizzare una particolare base di dati invia al sistema una richiesta di uso di tale database tramite la console di PariPari, utilizzando il comando “\ use” seguito dal nome del database da utilizzare e dalle credenziali definite per tale utente (il messaggio è del tipo “\ use dbname: username: password”). Il messaggio inserito dal sistema nella coda globale dei messaggi, viene recuperato da Message Delivery che lo smista al thread Command Interpreter in modo che possa interpretarlo. Tale processo recupera la richiesta di uso del database e le informazioni in esso contenute, come il nome del database da utilizzare, il nome e la password dell’utilizzatore, che vengono utilizzate successivamente per la procedura di login. Infatti il sistema prima di consentire all’utente di accedere al database richiesto deve verificare se questo ne sia autorizzato o meno: l’utente è autorizzato solamente nel caso che le sue credenziali corrispondano a quelle stabilite per tale database, ossia siano le stesse dell’utente che ha creato tale database. Command Interpreter effettua una ricerca attraverso il modulo DHT utilizzando

come chiave di ricerca il nome del database da utilizzare assieme allo username, in modo da ottenere gli indirizzi IP di tutti quei nodi che possiedono il corretto database da utilizzare e effettivamente creato precedentemente da un utente con lo stesso username di quello che lo vuole utilizzare attualmente. La ricerca può restituire due risultati significativi. Nel caso non restituisca alcun indirizzo allora significa che il sistema non contiene quel database richiesto e creato da quel particolare utente e dunque l'accesso viene negato in quanto il database risulta inesistente all'interno del sistema. Nel caso invece che restituisca una serie di indirizzi IP allora significa che il database è presente all'interno del sistema. A questo punto il thread verifica che tra i nodi restituiti dalla ricerca vi sia anche il nodo locale: in caso positivo allora si procede con una serie di verifiche, nel seguente modo. Prima di tutto viene effettuata la procedura di login in modo da verificare la correttezza delle credenziali passate dall'utente al momento della richiesta d'uso del database. Se la verifica è positiva allora l'utente risulta essere autorizzato all'utilizzo di tale database e dunque si passa alle verifiche successive che riguardano lo stato della connessione locale e del database da utilizzare. In particolare per prima cosa si controlla che all'interno del nodo locale non sia già presente una connessione con un database locale o remoto e successivamente si controlla che lo stato del database locale da utilizzare sia impostato a Ready. Nel caso vengano superati correttamente tutti questi controlli, il nodo locale imposta la propria connessione, corrispondente all'oggetto della classe Connected To, al database locale che si deve utilizzare. Nel caso che la prima verifica riguardante le credenziali passate non sia andata a buon fine allora il processo di accesso deve essere interrotto in quanto l'utente non risulta essere autorizzato a utilizzare tale database. Nel caso invece che il database locale da utilizzare non si trovi allo stato Ready e dunque non sia momentaneamente disponibile all'esecuzione delle operazioni SQL richieste oppure nel caso che il nodo abbia già stabilito una connessione precedentemente all'arrivo della nuova richiesta di connessione e dunque non sia momentaneamente disponibile per l'utilizzo del database richiesto, in entrambe le situazioni si procede all'esecuzione del protocollo di accesso remoto. Tale processo viene avviato anche nel caso che l'indirizzo IP del nodo locale non sia contenuto nella lista di indirizzi restituita dalla precedente ricerca su DHT in quanto il database che si vuole utilizzare è contenuto in calcolatori remoti e non localmente.

4.2.1 Protocollo di accesso remoto (RAP)

Il protocollo di accesso remoto si occupa di contattare un nodo in cui si trova il database da utilizzare, verificarne la disponibilità e in caso positivo di concederne l'utilizzo all'utente che lo

richiede. Tale procedura è realizzata dal thread denominato *Remote Access Protocol (RAP)*: esso è unico per ogni nodo in cui viene eseguito ed è dotato al suo interno di una coda di messaggi. Inizialmente Remote Access Protocol recupera la lista di indirizzi IP ottenuta dalla ricerca precedente su DHT di quei nodi che possiedono il database da utilizzare e contatta il primo nodo contenuto, inviandogli un messaggio di richiesta di accesso al database remoto da utilizzare contenete informazioni che riguardano il nome della base di dati e le credenziali dell'utilizzatore. Dopo di ché il thread imposta lo stato della connessione del nodo locale, l'oggetto Connected To, a *"Connecting"* e si mette in attesa della risposta. In questo stato eventuali comandi o operazioni SQL inviati dall'utente in locale e ulteriori richieste provenienti dalla rete di accesso a un database locale non vengono presi in considerazione. Il nodo contattato nel momento in cui riceve la richiesta lancia anch'esso il thread Remote Access Protocol che recupera a sua volta il messaggio di richiesta di accesso e le informazioni in esso contenute e effettua una serie di controlli. Per prima cosa controlla di possedere nella memoria di massa il database di cui si richiede l'utilizzo interrogando il database di supporto locale. In caso affermativo prosegue con la verifica delle credenziali passate avviando la procedura di login: se l'utente risulta autorizzato ad accedere al database remoto allora si procede con l'ulteriore verifica che nel nodo non siano già state stabilite altre connessioni a database locali o remoti e che il database da utilizzare sia allo stato Ready. Se tutti questi controlli vengono rispettati allora il nodo remoto invia al nodo richiedente un messaggio che indica che l'accesso remoto può avvenire, imposta la propria connessione, l'oggetto Connected To locale, allo stato *"Connected"* (connesso da un nodo remoto), memorizzandone il nome del database locale utilizzato, il nome dell'utente che lo ha richiesto e l'indirizzo IP del nodo da cui è giunta la richiesta e infine lancia il thread *Remote Query Manager (RQM)* per l'esecuzione delle operazioni, il cui funzionamento verrà definito nel prossimo paragrafo. Altrimenti nel caso anche uno solo di questi controlli non venga rispettato il nodo risponde al richiedente con un messaggio di rifiuto dell'accesso remoto indicandone la causa. Si prevede l'invio di un particolare messaggio di rifiuto nel caso l'utente non venga autorizzato all'accesso del database in modo da far terminare in entrambi i nodi il protocollo di accesso remoto. Una volta arrivata la risposta, il thread recupera il messaggio dalla propria coda interna che la contiene e la analizza: vi sono tre possibili situazioni. Nel caso il messaggio contenga una risposta di accesso consentito allora il thread imposta lo stato della connessione del nodo locale, l'oggetto Connected To locale, a *"Connected"* (connesso a un database remoto), memorizzandone il nome del database remoto utilizzato, il nome dell'utente che lo utilizza e l'indirizzo IP del nodo remoto in cui è contenuto il database utilizzato. Dopo di ché lancia anch'esso il thread Remote

Query Manager localmente per eseguire le operazioni SQL create dall'utente stesso e si mette in attesa dell'arrivo del messaggio di disconnessione da quel database. Nel caso invece che la coda di messaggi interna al thread RAP contiene un messaggio di accesso rifiutato allora esso recupera dalla lista degli indirizzi IP un nuovo nodo remoto contenente il database che interessa, lo contatta e poi esegue nuovamente la procedura descritta precedentemente. Si prosegue in questa maniera finché non viene trovato un nodo remoto disponibile ad accettare la richiesta di uso remoto: in tal caso la lista di indirizzi è stata scandita completamente e il protocollo di accesso remoto viene terminato negando all'utente di poter accedere al database richiesto. Il protocollo deve concludersi brutalmente anche nel caso che il thread richiedente riceva il messaggio di rifiuto particolare dovuto alla non autorizzazione dell'utente, che come ultima operazione deve impostare lo stato della connessione del nodo locale a *"Not Connected"*.

4.2.2 Gestore delle operazioni remote (RQM)

Il gestore delle operazioni remote garantisce la corretta esecuzione delle query SQL da effettuare sulla base di dati remota ed è realizzato attraverso un thread denominato Remote Query Manager. Anch'esso è dotato al proprio interno di una coda di messaggi e viene lanciato dal thread RAP in entrambi i nodi interessati dal protocollo di accesso remoto quando viene stabilita una connessione tra di loro.

L'utente dopo aver ottenuto l'accesso al database richiesto può finalmente interagire con esso inviandogli le operazioni SQL che deve eseguire. L'utente scrive la query e la invia al sistema che la racchiude all'interno di un messaggio e lo inserisce nella coda di messaggi globale. Message Delivery recupera il messaggio e controlla se è stata stabilita una connessione con un database locale o remoto all'interno del nodo: se l'indirizzo IP memorizzato nell'oggetto Connected To locale corrisponde all'indirizzo del nodo locale allora è stata stabilita in precedenza una connessione a un database locale altrimenti è stata stabilita una connessione a un database remoto. Nel caso la connessione sia locale Message Delivery smista la query da eseguire direttamente al thread SQL Interpreter, altrimenti la aggiunge alla coda di messaggi del thread RQM per effettuare la sua esecuzione in remoto. In quest'ultimo caso RQM recupera l'operazione dalla propria coda e la inserisce all'interno di un particolare messaggio da inviare al nodo remoto su cui deve essere eseguita, dopo di ché si mette in attesa della sua risposta per un tempo limitato. Il destinatario riceve il messaggio contenente la query SQL e lo inserisce nella coda del thread RQM locale. RQM lo analizza verificando che il messaggio sia giunto dal nodo che ha richiesto l'accesso remoto in precedenza: in tal caso il thread invia al mittente della

query un messaggio di conferma della sua ricezione. A questo punto il thread RQM del nodo mittente riceve la conferma e invia al destinatario un messaggio contenente un comando di avvio in modo da far partire l'esecuzione dell'operazione nel nodo remoto. Successivamente si mette nuovamente in attesa per un tempo limitato, ma in questo caso del risultato dell'operazione eseguita. Il thread RQM destinatario quando riceve il messaggio di avvio dell'esecuzione, recupera la query giunta precedentemente e la inserisce nella coda del thread SQL Interpreter in modo che venga eseguita localmente. SQL Interpreter procede come al solito, attivando i thread Transaction Synchronizer e Transaction Executor associati al corretto database locale da utilizzare in modo da garantire l'esecuzione di tale operazione e l'eventuale attivazione del processo di sincronizzazione. Transaction Synchronizer nel momento in cui va a creare la transazione contenente l'operazione stessa ne imposta il campo sender all'indirizzo IP del nodo che ha richiesto l'esecuzione di tale operazione in modo che quando Transaction Executor esegue tale transazione, con valore valido nel campo sender, ne recupera i risultati e li inserisce all'interno di un messaggio che poi viene aggiunto alla coda di messaggi interna al thread RQM locale. Il thread RQM non appena riceve il messaggio contenente il risultato dell'esecuzione dell'operazione lo invia al thread RQM del nodo richiedente, il quale lo riceve e a sua invia un messaggio di ritorno contenente la conferma dell'arrivo dei risultati. Dopo di ché il thread procede con l'invio di una nuova query.

Quella che è stata appena descritta è la procedura di esecuzione delle query in remoto nel caso tutto sia andato a buon fine: tuttavia può verificarsi una disconnessione del nodo richiedente o del nodo destinatario dell'esecuzione dell'operazione. Nel caso sia il nodo destinatario a disconnettersi durante questo processo allora si ha la scadenza di uno dei due timeout su cui era in attesa il thread richiedente, il quale si risveglia dall'attesa con la coda di messaggi vuota. In questo caso il thread recupera la query che dunque non è stata eseguita e risveglia il thread RAP che era in attesa di un'eventuale messaggio di disconnessione passandogliela. RAP si accorge di essere stato risvegliato da RQM e contatta un altro nodo per poter eseguire tale operazione. Anche il nodo mittente può disconnettersi dalla rete durante l'esecuzione della query in remoto: il nodo destinatario è dotato anch'esso di due timeout, il primo azionato nel momento in cui arriva la query da remoto in attesa del comando di avvio e il secondo dopo l'invio dei risultati al nodo richiedente in attesa della conferma del loro arrivo. La scadenza di uno dei due comporta la disconnessione del nodo richiedente dalla rete e la conseguente terminazione della procedura di esecuzione della query in remoto, e quindi del thread RQM locale, e della procedura di accesso remoto, e quindi del thread RAP locale.

4.2.3 Protocollo di disconnessione

Quando l'utente ha terminato di interagire con il database necessitato deve disconnettersi da esso. L'utente invia da console il comando "\ disconnect" per richiedere la disconnessione dal database utilizzato. Il sistema recupera il messaggio dalla coda globale di messaggi attraverso il thread Message Delivery che lo smista al thread Command Interpreter. A questo punto Command Interpreter verifica il tipo di connessione stabilita nel nodo locale, analizzando l'oggetto Connected To locale: se la connessione è stata stabilita con un database locale allora il thread termina la connessione impostando lo stato dell'oggetto Connected To a *"Not Connected"*, altrimenti se la connessione è stata stabilita con un database remoto allora si devono liberare dalle rispettive connessioni entrambi i nodi interessati precedentemente dall'accesso remoto. In particolare entrambe le connessioni dovranno essere impostate allo stato "Not Connected" e entrambi i thread RAP e RQM dovranno terminare la propria esecuzione in modo da ripartire nel momento in cui viene stabilita una nuova connessione remota. Per ottenere questi risultati non appena arriva il comando "\ disconnect" e la connessione stabilita è di tipo remoto, Command Interpreter risveglia il thread RAP locale che si era addormentato dopo che aveva lanciato il corrispondente thread RQM per l'esecuzione delle query in remoto. RAP riconosce di essere stato risvegliato da Command Interpreter, effettua una richiesta di terminazione dell'uso del database remoto, inviando un messaggio al nodo destinatario della connessione remota, e imposta lo stato dell'oggetto Connected To locale a *"Not Connected"*. Successivamente viene terminata l'esecuzione dei thread RQM e RAP locali. Il nodo remoto, in particolare il proprio thread RAP, quando riceve il messaggio contenente la richiesta di terminazione dell'uso del database remoto, verifica lo stato della propria connessione controllando che il messaggio sia giunto dal nodo a cui è effettivamente connesso in quell'istante. A questo punto viene terminata l'esecuzione del thread RQM locale e impostato lo stato dell'oggetto Connected To locale a *"Not Connected"*. Dopo di ché viene terminata l'esecuzione del thread RAP stesso. In questo modo ora entrambi i nodi non sono più connessi tra di loro e risultano disponibili per altre nuove connessioni.

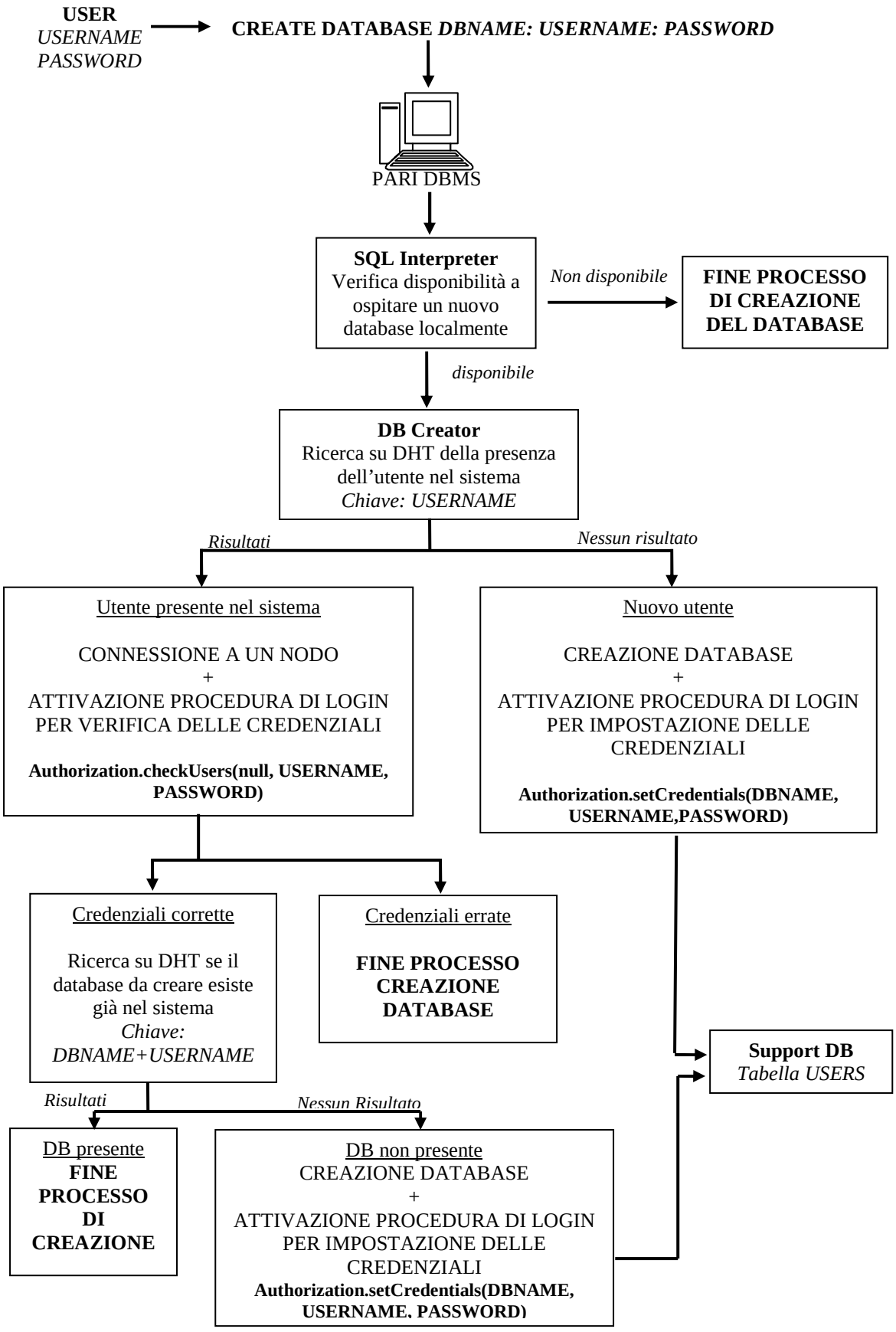
Conclusioni e sviluppi futuri

Con la stesura di questo elaborato si sono descritte le modifiche apportate al sistema Pari DBMS e le nuove funzionalità introdotte. Nella sua prima versione il plug-in garantiva solamente la creazione delle basi di dati e il loro corretto funzionamento attraverso i processi di sincronizzazione delle operazioni, di replicazione e di aggiornamento delle basi di dati. I cambiamenti attuati, attraverso le integrazioni con il modulo Diesel e con il plug-in Connectivity, hanno determinato un piccolo miglioramento dell'efficienza del plug-in diminuendo il carico di lavoro dei nodi e della rete stessa e permettendo il riutilizzo di alcune risorse necessarie al funzionamento del sistema. La nuova funzionalità di accesso remoto ha permesso di soddisfare uno dei requisiti fondamentali per il corretto funzionamento del sistema e già stabilito nella progettazione della prima versione di Pari DBMS: il poter accedere e usufruire delle basi di dati messe a disposizione del sistema da qualunque calcolatore connesso a PariPari, anche nel caso questo non disponesse nella propria memoria di massa dei dati necessitati. Con l'introduzione del login invece è stato soddisfatto uno dei requisiti fondamentali che caratterizzano ogni sistema di gestione di basi di dati, siano essi centralizzati che distribuiti: la privatezza dei dati, garantita attraverso il meccanismo di autorizzazione e controllo degli utenti. I cambiamenti tuttavia non si concludono in questa nuova versione del plug-in, che dovrà essere migliorata a sua volta portando a un livello di efficienza sempre più alto e garantendo nuove funzionalità nelle prossime versioni che verranno rilasciate. In futuro si prevede di integrare il plug-in con il modulo Local Storage per ottenere una gestione più efficace ed efficiente della memoria locale dei nodi e con il modulo Crediti per la gestione efficiente delle risorse esterne richieste dal plug-in per il suo corretto funzionamento. Inoltre quando verrà rilasciato il plug-in Login, ancora in fase di elaborazione, il sistema dovrà sostituire l'attuale processo di login usufruendo del servizio offerto dal nuovo modulo. Nelle prossime versioni si dovrà garantire che il plug-in goda del meccanismo di partizionamento delle basi di dati, requisito fondamentale stabilito nella prima progettazione di Pari DBMS ma molto complesso con il quale si garantirà efficientemente non solo la gestione di basi di piccole e medie dimensioni come avviene attualmente, ma di basi di ampie dimensioni.

Nel momento in cui Pari DBMS disporrà di tutte queste funzionalità allora potrà finalmente diventare un sistema di gestione di database affidabile ed efficiente, in grado di offrire ad un ampia gamma di utilizzatori gli stessi servizi alla pari di qualunque DBMS tradizionale.

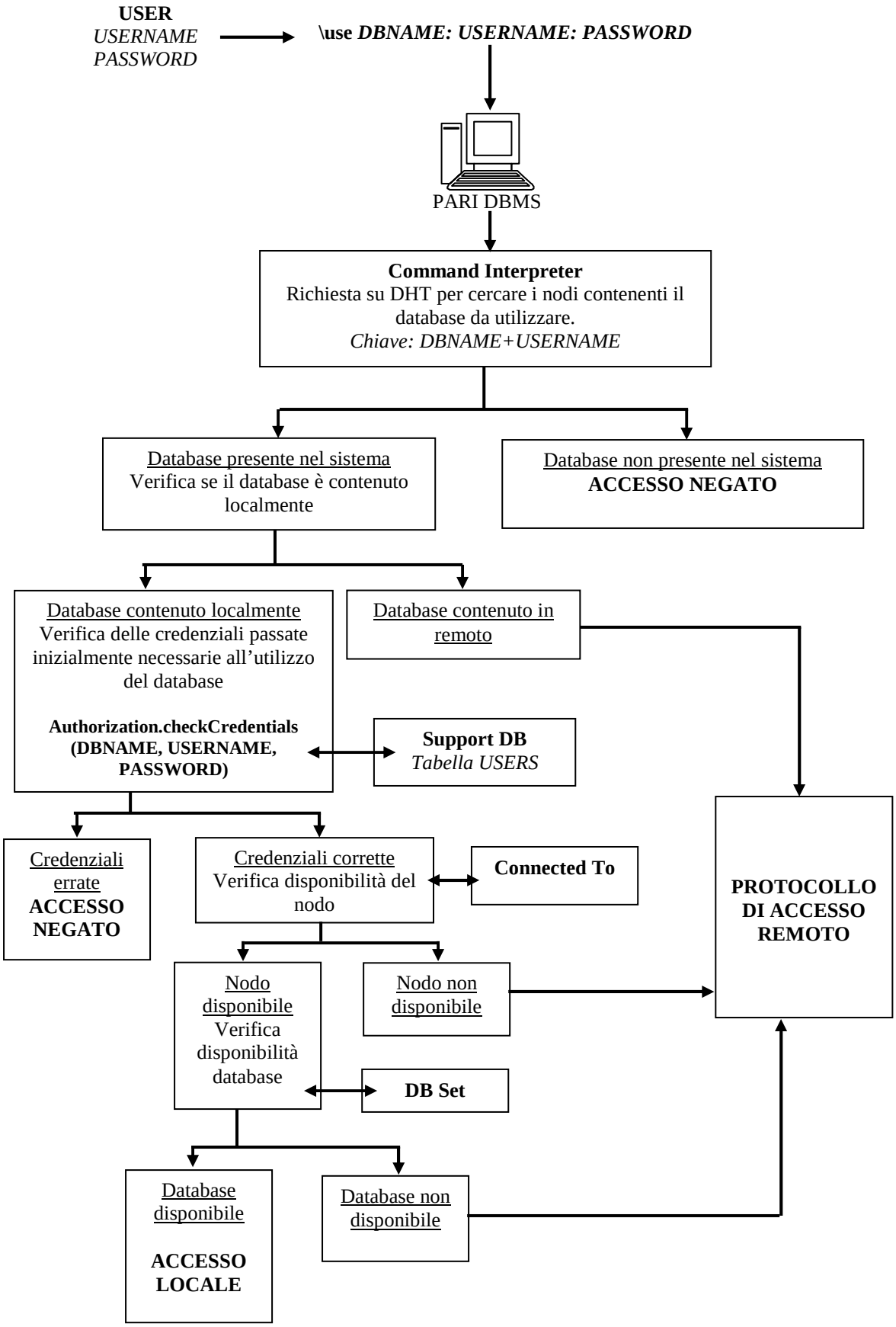
APPENDICE A

Schema a blocchi riguardante la procedura di login nel processo di creazione di una base di dati



APPENDICE B

Schema a blocchi del processo di accesso a un database.



Bibliografia

[1] Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Riccardo Torlone. *Basi di dati. Modelli e linguaggi di interrogazione.* Seconda edizione, McGrawHillItalia, 2006.

[2] Ramez A. Elmasri, Shamkant B. Navathe. *Sistemi di basi di dati – Fondamenti*. Quinta edizione, Pearson Addison Wesley, Paravia Bruno Mondadori Editori, 2007.

[3] Tesi di laurea triennale. *PARIPARI DBMS: (RE-)INIZIALIZZAZIONE E CHURN*. Jacopo Buriollo, A.A. 2007-2008.

[4] Tesi di laurea triennale. *PARIPARI DBMS: GARANZIE DI SERVIZIO*. Andrea Cecchinato, A.A. 2007-2008.

[5] Tesi di laurea triennale. *PariPari DBMS: sincronizzazione.* Alessandro Costa, A.A. 2007-2008.

[6] Sito ufficiale di HSQLDB, disponibile alla pagina web *http://www.hsqldb.org*.

[7] Sito internet *www.wikipedia.org*.

[8] Documentazione Java disponibile alla pagina web *http://java.sun.com/javase/6/docs/api*.