Reti di Calcolatori
Progetto: vendita biglietti palasport
Di Lazzari Matteo

1. Introduzione e ipotesi di lavoro

Il progetto vuole simulare la vendita di biglietti di un palasport.
L'applicazione, scritta in Java, descrive una realtà ridotta: quattro settori (A, B, C e D) di venti posti ciascuno.
Il lato client è usato dall'operatore al botteghino per verificare la disponibilità ed impegnare i posti.
Esso dispone di interfaccia grafica molto intuitiva, che richiede l'immissione di user name, password, settore, fila e posto, restituendo l'eventuale disponibilità.
Sarà l'operatore umano a rilasciare il biglietto (che potrebbe essere stampato dall'applicazione stessa) e a verificare il pagamento sul posto.

L'affidabilità e la continuità del servizio è garantita dalla replicazione: sono previsti due server entrambi attivi ma su nodi diversi, con lo stesso applicativo.
E' previsto il guasto di uno solo dei due.
La risposta affermativa è data al client solo dopo una fase di coordinazione fra i due server.

La disponibilità di posti è divisa a metà fra i due server (A-B e C-D), che saranno *master* per la metà che gli compete e *slave* per l'altra (ovviamente in modo duale).
Il client "conosce" entrambi e si rivolge al master proprietario del posto desiderato.
Questa organizzazione garantisce distribuzione del carico sui due server ed evita casi di starvation possibili in una soluzione con un server master unico (la soluzione è più *fair*).

E' previsto anche il caso di recovery del client.

Le ipotesi di lavoro sono:

a) Guasto singolo per i server: oltre all'interruzione di servizio, al momento della ripartenza, in caso di guasto su entrambi i server, si avrebbe il problema di individuare quale dei due database è quello più recente;
b) Si suppone che non avvengano guasti sulla linea di comunicazione fra i due server, perché consentirebbe ad entrambi di lavorare senza coordinazione;
c) La coordinazione fra i due server avviene singolarmente per ogni richiesta, si suppone di avere traffico contenuto per rendere valida la soluzione adottata;
d) Consistenza dei due database server (l'applicazione rileva questo caso sui server, non conferma il posto al client, ma prosegue nell'esecuzione);
e) Ipotesi di bassa sicurezza: le richieste client sono accompagnate da user name e password distribuite in modo esterno, e trasmesse in chiaro sulla rete;
f) E' ovvio che l'ambiente server deve essere sicuro, altrimenti chiunque potrebbe modificare il database e la lista delle password;
g) Richieste singole.

2. Descrizione degli elementi del sistema

2.1 Client

Il client "conosce" entrambi e si rivolge al master proprietario del posto desiderato; se quest'ultimo non risponde la trasmissione si sposta sull'altro.
L'unico momento in cui i client accedono random ai server è quello di aggiornamento all'inizializzazione.

Il client prevede l'invio di tre tipi di messaggi:

- **AGG**: aggiornamento completo del database locale (una tantum all'inizializzazione);
- **RIC**: richiesta per un posto;
- **VER**: verifica di un posto, porta all'occupazione del posto se è libero;

All'inizializzazione il client invia la richiesta di aggiornamento completo del database locale.

Sulla base del database locale viene inviata la richiesta (non parte se il posto è già occupato in loco).
Tale richiesta è singola per semplicità.
Se il posto è trovato occupato sul server, si ottiene l'aggiornamento completo del settore richiamato.

In caso di caduta del server destinatario, il client si rivolge automaticamente all'altro ed effettua la verifica del posto, in quanto la procedura del primo server potrebbe essere arrivata alla fase di coordinazione ed aver occupato il posto anche nel secondo; il confronto fra i campi user e count associati al posto e alla richiesta corrente consente di concludere l'operazione.

In caso di caduta del client la procedura tenta di continuare sui server. Al riavvio del client viene verificata la presenza del salvataggio su file (creato all'inizio della richiesta e distrutto alla risposta) ed eventualmente viene lanciata la verifica (che può portare all'occupazione del posto se libero).

Il client apre e chiude la connessione ad ogni operazione, non occupa risorse inutilmente.

2.2 Server

Si è scelto di aprire due ServerSocket:

- una per l'ascolto dei client;
- una per l'ascolto dell'altro server.

Cronologicamente la fase di inizializzazione consiste:

- inizializzazione dell'interfaccia grafica;
- tentativo di aprire una connessione verso l'altro server:
  - se riesce aggiorno il database da questo;
  - altrimenti aggiorno il database da file. ATTENZIONE: questa operazione potrebbe causare la riemissione di biglietti già venduti se quello che si avvia per primo non ha i file più aggiornati.
- Apertura della socket d'ascolto per l'altro server;
- Inizializzazione del database users ammessi al servizio;
- Apertura socket d'ascolto per i client.

La coordinazione fra i due server è gestita con due tipi di messaggi:

- **START**: con cui il server comunica la propria attivazione all'altro e ne richiede il download del database;
- **ORD**: Il server ordina all'altro di modificare lo stato di un posto appartenente alla metà in cui è master.

Ognuno dei due server è master nella propria metà di posti e slave nell'altra, o meglio, se tutto è on, un server riceverà richieste per una metà dei posti dai client e per l'altra metà dall'altro server.

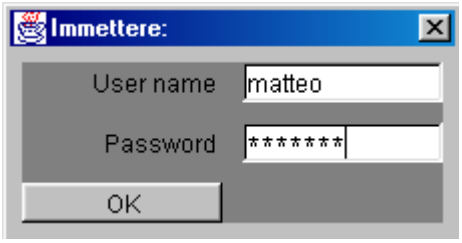
L'applicazione è progettata in modo da tollerare il guasto di uno dei due server. Nel tempo in cui uno è down, l'altro svolge il lavoro da solo.

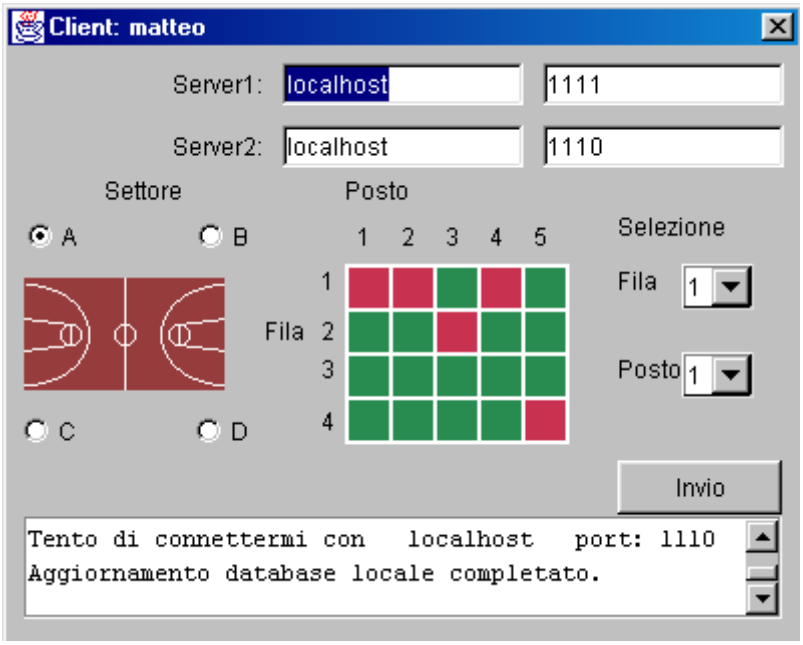
3. Istruzioni

Il sistema è dotato di interfaccia grafica molto intuitiva.

3.1 Client

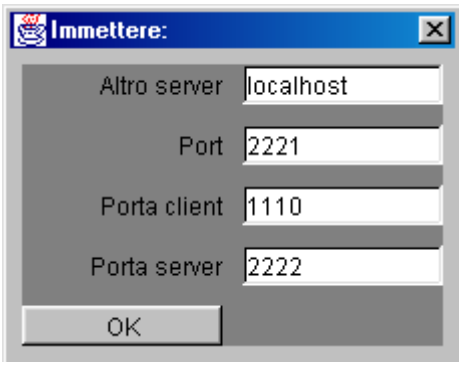
Va attivato quando almeno un server è attivo, altrimenti non trovando l'aggiornamento dei posti disponibili, termina automaticamente.

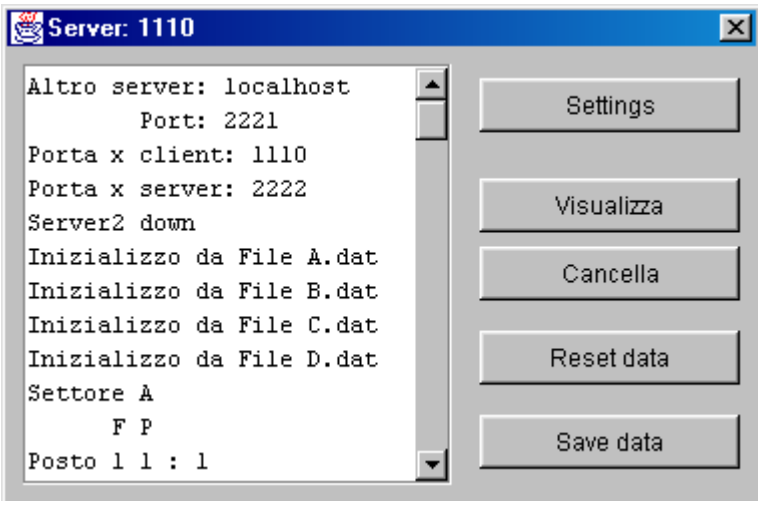




Una finestra di dialogo chiede di immettere username e password per la sessione di lavoro. L'applicazione client non effettua nessun controllo localmente sulla loro validità, sarà compito del server.
La parte superiore è di servizio, specifica i parametri per la comunicazione con i due server.
Il canvas centrale illustra la disponibilità di posti (rosso = occupato, verde = libero).
Con il pulsante "Invio" si richiede ad un server di impegnare il posto correntemente specificato sull'interfaccia.
La parte inferiore restituisce informazioni riguardo all'andamento e il risultato dell'operazione.

3.2 Server





Devono essere avviate due esecuzioni del programma con i parametri di comunicazione (richiesti all'avvio in una finestra di dialogo) opportunamente duali.
L'interfaccia del server è prevalentemente passiva (ampia area di testo dove vengono visualizzate le operazioni compiute), ma dotata di alcuni pulsanti:

- **Settings**: consente di cambiare i parametri di comunicazione (mostra nuovamente la finestra di dialogo);
- **Visualizza**: mostra a video lo status complessivo dei posti (potrebbe essere associato ad una stampante);
- **Cancella**: delete di tutto quello che c'è nell'area di testo;
- **Reset data**: consente di rendere nuovamente disponibili tutti i posti, è un pulsante che dovrà essere usato solo quando nessun client è in esecuzione, perché quest'ultimi non vedrebbero i posti rilasciati;
- **Save data**: effettua il salvataggio dei dati correnti su disco; è un'operazione che va eseguita solo allo spegnimento e nello stesso momento per entrambi i server, perché al successivo riavvio, il server con il database più "vecchio" potrebbe rendere disponibili posti già venduti (eventualmente si potrebbero scambiare i dati alla fine della giornata di lavoro).

OSS: Non ho implementato alcun salvataggio automatico su disco da parte dei server perché per l'ipotesi di guasto singolo non è necessario (al riavvio scarico il database aggiornato dall'altro).
Con "Save data" posso effettuare il salvataggio alla fine della giornata di lavoro; tale salvataggio deve essere lo stesso per entrambi i server, altrimenti se al riavvio si attivasse il server con il database meno aggiornato, si potrebbe vendere due volte lo stesso posto.

4. Prova dell'applicazione

L'applicazione è stata provata su una macchina Windows 98 aprendo quattro finestre separate (due server e due client) e lavorando in locale (specificando *local host* come indirizzo).

E’ inoltre stata effettuata la prova sulle macchine Windows NT del Lab2, lanciando i due server su macchine distinte.
Non si è verificato alcun significativo peggioramento delle prestazioni (ovviamente il carico di dati trasmessi e le distanze fra i “nodi” non sono elevate).

5. Note conclusive

Il progetto si inserisce nel programma di Reti di Calcolatori alla voce Comunicazione Client-Server, ponendo l’accento sulla coordinazione di risorse replicate (due server) e recovery di tutte le entità partecipanti (sia client, sia server).
I due server sono entrambi attivi e paralleli, garantiscono fairness, continuità di servizio (per guasto singolo) e un minimo di sicurezza (l’adozione di password è solo esemplificativa, il sistema è facilmente attaccabile).
Lo stato della comunicazione è dalla parte del client, che si incarica del completamento dell’operazione (ridirigendo la comunicazione ed effettuando verifica) in caso di crash di qualsiasi entità del sistema. In questo modo viene ridotto il carico di lavoro sul server e snellita la sua procedura di ripartenza.

6. Descrizione classi

Client

MioClient: main e generazione dell’interfaccia grafica.
Campo: Canvas dell’interfaccia grafica; è generati da CreaCampo che è un’applet per comodità in fase di creazione (visualizzazione al di fuori dell’applicazione).
Quadro: come Campo; generata da CreaQuadro.
finPass: finestra di dialogo per l’immissione di username e password.
GestoreClient: si occupa della gestione delle entità interattive dell’interfaccia.
InvioClient: Thread per la gestione di qualsiasi tipo di invio del client.
CurrentSel: classe per la memorizzazione della selezione corrente.

Server

MioServer: main e generazione dell’interfaccia grafica.
finServ: finestra di dialogo per l’immissione dei parametri di comunicazione.
finRes: finestra di dialogo per la conferma del reset.
GestoreServer: si occupa della gestione delle entità interattive dell’interfaccia.
InAscolto: Thread per la creazione di una ServerSocket di ascolto.
GestisciRicezione: Thread per la gestione delle richieste dei client (una implementazione per ogni connessione; il server è parallelo).
CoordinazioneServer: Thread per la gestione delle richieste dell’altro server.
Sector: classe synchronized che contiene le informazioni relative ai posti (una istanza per settore per non sequenzializzare troppo l’accesso al database).