Relazione Assignment 1

Programmazione Concorrente e Distribuita

Mariano Sciacco (1142498)

2018 — 2019

Corso di Laurea in Informatica, Scuola di Scienze, Dipartimento di Matematica, UniPD

MerkleTree

Introduzione

Il primo assignment richiede la realizzazione di una struttura dati `HashMultiSet`, che tiene traccia delle frequenze di oggetti con la stessa chiave nella struttura dati, e la realizzazione di un merkleClient (con un relativo server) per simulare un accertamento di validità (*validity proof*) delle transazioni attraverso il controllo degli hash basato sul modello **merkleTree**.

Realizzazione del HashMultiSet

La struttura dati `HashMultiSet<T,V>` è stata realizzata implementando due array che verranno rispettivamente usati per:

- Tenere traccia degli oggetti inseriti
- Associare una frequenza relativa all'oggetto inserito

Trattandosi di due array, si utilizzerà il medesimo **index** in modo parallelo per gestire le due strutture dati di supporto. L'HashMultiSet implementa inoltre anche i metodi per eseguirne la costruzione e la linearizzazione dei dati, come riportato sui relativi test. Ovviamente le eccezioni sull'esistenza del source file vengono prontamente effettuati, basandosi principalmente sulle eccezioni che vogliono essere sollevate.

L'aggiunta di un elemento comporta la conversione esplicita al tipo `(V)` che compone questa struttura dati, il cui tipo è sicuramente un estensione del tipo *Numbers*, visto e considerato che le frequenze devono essere numeriche.

Funzionamento del MerkleClient

Il client tiene pronte le transazioni che vuole verificare ed esegue i seguenti passi, una volta invocato il metodo che **checkWhichTransactionValid()**:

1. Vengono inizializzate le strutture dati di supporto per salvare i risultati
2. Per ciascun elemento da verificare viene aperta una connessione di rete tramite `Socket`.
3. Si inizializzano i relativi buffer per leggere gli stream I/O del socket
4. Il **client** invia al **server** la richiesta per la transazione ID presa a carico in quel momento.
5. Il **client** riceve gli **hash** dal **server** e li salva in una lista.
6. In conclusione, esegue la verifica degli hash e riporta i risultati in un array, chiudendo il socket e i relativi buffer.

Il client implementa le relative eccezioni per I/O e per l'impossibilità di connessione. Per la connessione si fa uso di un `PrintWriter` e di un `ObjectInputStream` con cui è possibile inviare stringhe in uscita e ricevere in ingresso direttamente delle liste di elementi (previo cast esplicito).

Funzionamento del MerkleServer

Il server implementato è molto semplice ed esegue solamente un invio degli hash richiesti, senza effettuare controlli sulla strutture dati di un merkleTree. I passi di funzionamento sono i seguenti:

1. All'avvio, rimane in attesa di connessioni e si predispone all'ascolto con il relativo indirizzo IP e porta.
2. Il **server** legge la richiesta di connessione e manda gli **hash** da far controllare al **client** (che per questo esercizio sono fittizzi).
3. Se tutto è andato a buon fine, chiude la connessione e rimane nuovamente disponibile per altre richieste.

Anche il server, come il client, implementa le eccezioni per mancata ricezione di una transazione o per tentativo di connessione fallito.

Risultati

Eseguendo i due programmi, si otterranno come risultati le comunicazioni tra client-server e il relativo risultato di validazione delle transazioni. In tale caso entrambi gli hash da analizzare saranno errati e dunque entrambe le transazioni invalide.

Nota: è stato rallentato il processo di risposta del server con un cooldown di 2 secondi a richiesta così da poter vedere i risultati venire computati.

Print del server

```
[Server] Avvio... server in attesa di connessioni.
[Server] Connessione ricevuta: /127.0.0.1:49548
[Server] <- TransID ricevuto: 0ff89de99d4a8f4b04cb162bcb5740cf
[Server] -> Hash per il confronto inviati
[Server] Ok... server in attesa di connessioni.
[Server] Connessione ricevuta: /127.0.0.1:49550
[Server] <- TransID ricevuto: 8ca10608a248910c25083c3dab4371c3
[Server] -> Hash per il confronto inviati
[Server] Ok... server in attesa di connessioni.
```

Print del client

```
[Client] Connessione al server [127.0.0.1:2323]...
[Client]  -> Invio TransID: 0ff89de99d4a8f4b04cb162bcb5740cf
[Client] <- Ricezione Hash: [c95c48a8ba50577ebee763b533790f90,
                             4cb427c11739339dae985d12bac8b5e8,
                             52d2b3833601fb9c970a9a12cf7f2db6]
[Client] Calcolo il risultato...
[Client] ==> Transazione non valida!
[Client]  -> Invio TransID: 8ca10608a248910c25083c3dab4371c3
```

```
[Client] <- Ricezione Hash: [c95c48a8ba50577ebee763b533790f90,
                             4cb427c11739339dae985d12bac8b5e8,
                             52d2b3833601fb9c970a9a12cf7f2db6]
[Client] Calcolo il risultato...
[Client] ==> Transazione non valida!
[]
```