REQUEST PIPELINE

Request Pipeline

- Intento: Migliorare la latenza mandando richieste multiple sulla connessione senza aspettare le risposte alle precedenti richieste
- Problema
  - La comunicazione fra server di un cluster usando un singolo canale può causare problemi di prestazioni se le richieste devono aspettare le risposte delle precedenti richieste
  - Per ottenere un migliore throughput, la coda di richieste sul server dovrebbe essere sufficientemente piena per far sì che la capacità del server sia pienamente usata
  - Se si manda solo una richiesta alla volta molta della capacità del server sarà sprecata

Request Pipeline

- Soluzione
  - I nodi mandano le richieste agli altri nodi senza aspettare le risposte alle precedenti richieste
  - Per questo, si usano due thread separati uno per mandare le richieste su un canale di rete e un altro thread per ricevere le risposte dal canale di rete
  - Il nodo richiedente manda le richieste senza aspettare la risposta
  - Il nodo ricevente elabora immediatamente la risposta o la mette in una coda

Request Pipeline

- Possibili problemi
  - Il nodo che accetta le richieste potrebbe essere “saturato”. Per questo motivo si mette un limite al numero di richieste inevase
  - Ogni nodo può mandare un massimo di richieste agli altri nodi. Al raggiungimento del limite, senza aver ricevuto risposte, il richiedente è inibito dal mandare richieste
  - Una coda può essere tenuta per le richieste non inviate. Una volta che una risposta viene ricevuta, questa libera una possibile richiesta che era in coda
  - La gestione dei fallimenti può essere difficile. Una prima richiesta che fallisce, potrebbe essere ritentata e una seconda richiesta potrebbe essere eseguita prima di ritentare la prima richiesta
  - Occorre un meccanismo per rifiutare le richieste fuori ordine. Un identificativo unico per richiesta, fornito dal lato server, permette di controllare l’ordine delle richieste

Esempio

- Un servizio remoto (`weatherService`) accetta una richiesta (istanza di `Request`) e fornisce una risposta (istanza di `Response`)

```
public interface Service extends Remote {
    Response weatherService(Request request) throws RemoteException;
}
```

- Il chiamante (lato client) non si blocca al momento della chiamata RMI al servizio e prenderà il risultato successivamente

```
Request request = new Request("req-1", "Honolulu");
var response = CompletableFuture.supplyAsync(() -> sendRequest(request));

private Response sendRequest(Request request) {
  try {
     return service.weatherService(request);
  } catch (RemoteException e) { return null; }
}
```

- Il thread fornito da `supplyAsync` esegue la chiamata remota e aspetta, mentre il thread che esegue il client prenderà il risultato una volta presente sulla variabile `response`