Simulazione di un sistema P2P di scambio risorse

Programmazione concorrente e distribuita
Progetto A.A. 2013/2014

1 Descrizione del progetto

Il progetto consiste nella realizzazione di un sistema P2P per la condivisione e lo scambio di un certo numero di risorse.

Il sistema è costituito da un certo numero di Server e da un certo numero di Client che si connettono e si disconnettono dal sistema. Ogni risorsa R_K è caratterizzata da un nome, R, e da un numero K di parti in cui è scomposta. Ogni client ha un certo numero di risorse ed ha una certa capacità di download D che gli permette di scaricare fino a D parti di risorse contemporaneamente.

Un client C che si connette al sistema mette a disposizione l'insieme di risorse che possiede. Un client C connesso può richiedere al sistema una risorsa R_K di cui ha bisogno; il server S a cui C è connesso dovrà indicare a C l'elenco dei client che sono attualmente connessi al sistema e in possesso di una copia di R_K. C quindi comunicherà direttamente con un certo numero di client di questo elenco, scaricando concorrentemente più parti di R_K da diversi clienti, in modo da massimizzare l'utilizzo della propria capacità di download. Se durante la fase di scaricamento di una parte di R, un cliente contattato si disconnette dal sistema, C deve poter reindirizzare la richiesta di quella parte ad un altro dei client che la possiedono. Si suppone che un client abbia infinita capacità di upload, cioè può servire le richieste di infinite parti di risorsa purché ogni richiesta arrivi da un client diverso. Perciò, dato un client A che richiede una risorsa divisa in 3 parti e un client B che è l'unico a possedere quella risorsa, non è possibile che A apra 3 connessioni con B per scaricare contemporaneamente le 3 parti della risorsa; A dovrà scaricare da B la risorsa connettendosi con B per 3 volte in modo sequenziale.

Al fine di rendere la simulazione verificabile ogni scaricamento di una parte deve essere simulato con un tempo di attesa costante per tutti i client *ma configurabile in modo agevole a livello di codice sorgente*.

Il programma deve gestire richieste e scambi concorrenti di diverse risorse. L'applicazione inoltre deve essere distribuita, cioè i vari client e server devono essere rappresentati da programmi distinti che possono risiedere su JVM distinte.

2 Requisiti del progetto

Si chiede di sviluppare un progetto che rispetti i requisiti elencati nel seguito. Le funzionalità non specificate in questo elenco possono essere implementate a piacere, tenendo presente che la valutazione finale intende **privilegiare la chiarezza della struttura e del codice** dell'applicazione piuttosto che le sue funzionalità aggiuntive.

2.1 Funzionalità

Di seguito si riportano le funzionalità che ci si aspetta siano implementate per ciascuna entità.

Ogni oggetto *Server*:

- è identificato da un nome;
- mantiene la lista di client che sono registrati presso di lui e che sono attualmente connessi al sistema;
- mantiene l'elenco di risorse che i client a lui registrati attualmente possiedono. In particolare, i server non possiedono alcuna risorsa;
- mantiene l'elenco degli altri server che compongono il sistema.

Ogni oggetto *Client*:

- è identificato da un nome ed è caratterizzato da un numero intero positivo che rappresenta la capacità di download D;
- possiede un insieme di risorse. Ogni risorsa è identificata da un nome e da un numero intero positivo che rappresenta il numero di parti (cioè se due risorse del sistema hanno lo stesso nome ma differenti parti sono da considerare risorse diverse perché identificate da dati diversi);
- si può connettere e disconnettere ad uno sei server del sistema;
- può richiedere al sistema una risorsa R_K; la richiesta va gestita nel modo seguente:
  - il client C richiede al server S a cui è registrato l'elenco dei N client attualmente attivi che possiedono R_K. Per identificare tale elenco, S deve ricercare tra i clienti registrati presso di lui o presso uno degli altri server del sistema;
  - il client C deve scaricare concorrentemente le K parti della risorsa in modo da (i) **massimizzare l'uso della capacità di download** e (ii) in modo che **diverse parti possono essere scaricate da uno stesso client solo sequenzialmente**.
    NOTA: supponendo che ad un certo istante di tempo la capacità di download sia D' (cioè $D - d$ con d il numero di download in corso), che manchino K' parti (cioè $K - k$ con k il numero di parti scaricate) e che ci siano N' client disponibili per quella risorsa ($N - n$ con n il numero di client già impegnati con C che hanno la risorsa R), allora C apre $min(D', K', N')$ connessioni con i client.
  - una volta ottenuta la risorsa R_K, C deve metterla a disposizione di altri clienti del sistema comunicando al server S l'avvenuto aggiornamento delle proprie risorse.
- Ogni client tiene traccia, per ognuna delle proprie parti di risorsa, di quanti e quali client hanno scaricato quella risorsa.

2.2 Interfaccia grafica

L'interfaccia grafica deve essere pulita e chiara in modo da facilitare la lettura dello stato corrente del sistema (e la correzione del progetto).

La figura rappresenta un esempio di interfaccia grafica. In questa GUI mentre i log sono delle semplici aree di testo a cui vanno appesi i vari messaggi degli eventi, gli altri riquadri dovrebbero essere delle liste in modo che ogni riga sia legata a un determinato oggetto. Nel riquadro "Coda download", ad esempio, dato l'oggetto 'A:1' (parte 1 del file A), gli eventi ad esso legati ('[in corso]' e '[completato]') dovrebbero apparire nella stessa riga.

Si noti che per cercare un file è necessario digitare sia il nome sia il numero di parti.

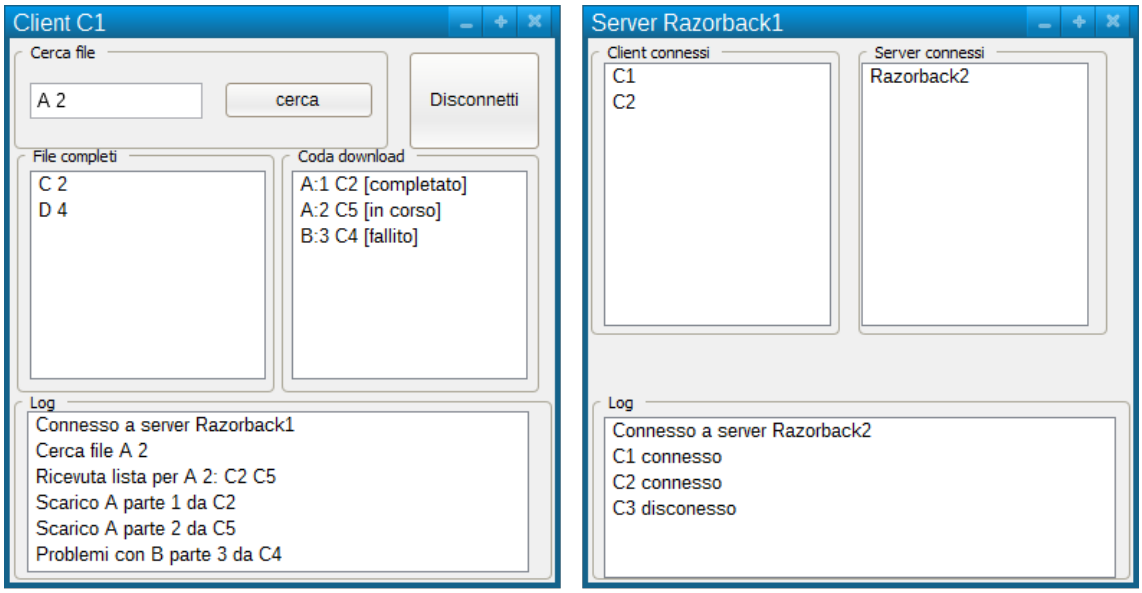

Figure 1: Esempio di GUI di un Client (a sinistra) e di un Server (a destra).

2.3 Concorrenza

- Un singolo client richiede una risorsa per volta, ma deve essere in grado di gestire concorrentemente la richiesta di una risorsa e la fornitura ad altri client delle risorse di cui è in possesso.
- Il server deve essere in grado di gestire concorrentemente richieste di connessione, disconnessione e richieste di risorse da diversi client.
- L'intero sistema deve essere in grado di gestire richieste, scambi di risorse, connessioni e disconnessioni di clienti in modo concorrente.

2.4 Distribuzione

- Le parti di risorsa devono essere scambiate solo tra client, mentre nelle comunicazioni client-server e server-server devono essere passate solamente informazioni sulla localizzazione delle risorse.
- Tutte le comunicazioni remote devono utilizzare il modello RMI.
- I riferimenti remoti agli oggetti Client non possono essere inseriti in alcun registro pubblico RMI.
- Il programma deve essere robusto, deve cioè gestire in modo semplice ma opportuno eventuali fallimenti delle entità del sistema (se un client si disconnette i client collegati devono mostrare un messaggio di errore e provare a scaricare da un altro client; se un server si disconnette un client deve mostrare che non può effettuare la ricerca, o non si può connettere per registrarsi, ecc.).

3 Valutazione del progetto

Un buon progetto dovrà essere sviluppato seguendo i principi basilari della programmazione ad oggetti in Java: encapsulation e information hiding, modularizzazione del codice, riutilizzo del codice, polimorfismo, classi astratte/interne/interfacce, gestione della concorrenza, robustezza, etc, anche per quanto concerne l'interfaccia grafica.
La valutazione del progetto terrà conto

- dell'aderenza del progetto ai precedenti principi,
- della chiarezza del codice e della struttura dell'applicazione,
- della corretezza del codice concorrente e della robustezza del codice distribuito,
- della **capacità di risolvere il problema assegnato in modo semplice ma esauriente**. Sono quindi preferibili i progetti che implementano in modo semplice e chiaro una soluzione corretta, mentre viene **SCORAGGIATA l'aggiunta di funzionalità e aspetti grafici non richiesti dalla presente specifica**.

4 Regole per la consegna del progetto

Il progetto dovrà essere realizzato da ogni singolo studente in modo INDIPENDENTE.

4.1 Come verrà effettuato il test operativo del progetto

Per quanto riguarda la valutazione del progetto, questo verrà eseguito localmente su un sistema operativo Linux, dove vi è una installazione di Java 7.
Al fine di standardizzare la procedura di valutazione i vari main dovranno aderire alla seguente specifica:

- il main del Server accetterà come parametro il nome del server. Esempio:
  `java server.Server Razorback`
- il main del Client accetterà il nome del Client, il nome del Server al quale vuole connettersi, un numero intero positivo come capacità di download e una serie di stringhe alternate da numeri interi positivi per definire l'insieme delle risorse e relative parti. Esempio di cliente di nome client1 con 3 risorse (J, K, L di 2, 7, 8 parti rispettivamente), capacità di download 5 che si vuole connettere a Razorback:
  `java client.Client client1 Razorback 5 J 2 K 7 L 8`
- Al fine di rendere la simulazione verificabile ogni scaricamento di una parte deve essere simulato con un tempo di attesa costante per tutti i client *ma configurabile in modo agevole a livello di codice sorgente*.

Sarà necessario creare un `Makefile` con almeno due regole:

- la regola `default` compila l'intero progetto;
- la regola `start` esegue una serie di comandi per far partire i vari main.
- la regola `stop` chiude i processi delle varie JVM

È FORTEMENTE CONSIGLIATO usare il Makefile proposto come base di partenza.
ATTENZIONE: errori occorsi durante l'esecuzione dei passi precedenti (dovuti a file mancanti o script non funzionanti), comporteranno una valutazione insufficiente del progetto.

4.2 Cosa consegnare

Alla luce della procedura di test illustrata sopra, la root del pacchetto che si dovrà consegnare, dovrà contenere il file `Makefile`.

Inoltre, al fine di permettere la valutazione del progetto è **OBBLIGATORIO** consengare una **breve relazione** che illusri le scelte progettuali effettuate. Dovranno essere illustrati brevemente i seguenti aspetti:

1. architettura del programma, mettendo in luce le scelte implementative più significative;
2. gestione corretta della concorrenza e della distribuzione, in particolare.

Tale presentazione non dovrà superare le 3 pagine (rispettando misure accettabili per quanto riguarda margini e dimensione del carattere). Nel caso venga presentato un documento con un numero maggiore di pagine, quelle successive alla terza non verranno considerate.

4.3 Come e quando consegnare

Il progetto va consegnato dalle macchine del laboratorio invocando il comando

```
consegna programmazione3-13-14
```

dalla directory contenente i file da consegnare. **Non** saranno accettate altre modalità di consegna (ad es. via email). È possibile consegnare remotamente il progetto usando il server `ssh.studenti.math.unipd.it`.

Le date di consegna del progetto saranno indicate sul sito del corso. Per dettagli sulle modalità d'esame si rimanda alle informazioni contenute nelle regole del corso.

4.4 Esempio di Makefile

Supponendo che:

- il main del Server sia chiamato `ServerStarter` ed sia definito in un package `server`;
- il main del Client sia chiamato `ClientStarter` ed sia definito in un package `client`;

Il `Makefile` può essere creato come segue:

```
# Makefile da inserire nella root del progetto: la cartella
# che contiene le cartelle dei vari package "server", "client".

JC = javac
.SUFFIXES: .java .class
.java.class:
        $(JC) $*.java

CLASSES = \
        server/ServerStarter.java \
        client/ClientStarter.java

default: classes

classes: $(CLASSES:.java=.class)

clean:
```

```
	$(RM) server/*.class
	$(RM) client/*.class

start:
	rmiregistry &
	sleep 2
	xterm -e "java server.ServerStarter Razorback1" &
	xterm -e "java server.ServerStarter Razorback2" &
	sleep 2
	xterm -e "java client.ClientStarter C1 Razorback1 3 A 1 B 4 C 6" &
	xterm -e "java client.ClientStarter C2 Razorback2 3 A 1 D 2" &
	xterm -e "java client.ClientStarter C3 Razorback2 3 E 2" &

stop:
	killall -q rmiregistry &
	sleep 1
	killall -q xterm &
```

Per evitare errori legati al rmiregistry è consigliabile che l'esecuzione del comando "rmiregistry &" avvenga nella root del progetto.