Java Server farm

M. Danelutto

Progetto conclusivo LPRb A.A. 2006-2007

Versione 1.0

1 Server farm

Lo scopo del progetto é la realizzazione di un server farm (vedi la definizione di server farm di Wikipedia alla figura 1) dedicato al calcolo, cioé di un server distribuito in grado di calcolare, in base alle richieste dei clienti, una serie di task $\langle x_1, f_1\rangle, \ldots, \langle x_m, f_m\rangle$ per ottenere una serie di risultati $f_1(x_1), \ldots, f_m(x_m)$ Gli utenti possono connettersi al server farm come ad un normale server e mediante un protocollo opportuno possono comunicare al server un certo numero di coppie funzione/dati $\langle x_i, f_i\rangle$. In conseguenza di tale richiesta riceveranno i risultati $f_1(x_1), \ldots, f_m(x_m)$ secondo le modalitá del protocollo descritto alla sezione 2. Il calcolo dei risultati avviene, nel server farm, utilizzando una serie di macchine opportunamente configurate, ognuna delle quali sará in grado di eseguire uno qualunque dei task proposti dall'utente.
Il progetto si divide logicamente in tre parti:

1. la realizzazione del server vero e proprio, che interfaccia il server farm con l'utente implementando un ben preciso protocollo,
2. la realizzazione del codice relativo al singolo nodo della server farm, cioé di uno dei nodi che, su richiesta del server del punto a) calcola effettivamente i task dell'utente
3. la realizzazione di clienti che permettano il test delle funzionalitá del sistema.

2 Protocollo Server

Client e server operano utilizzando un protocollo che prevede l'invio di una richiesta di servizio da parte del cliente e, di conseguenza, l'invio di un pacchetto di risposta da parte del server. Il protocollo é un protocollo ASCII (ovvero tutti i pacchetti di richiesta e risposta sono costituiti da caretteri ASCII) ma é possibile che vengano scambiati anche pacchetti non ASCII per determinate funzionalitá che comunque non coinvolgono i messaggi di richiesta e di risposta.

Il server riceve messaggi di richiesta sulla porta

`serverfarm.ServerFarm.SERVERFARMPORT`

utilizzando TCP. Il client puó inviare al server una richiesta nel formato:

```
<ReqMsg> ::= SERVERFARM <ReqOp> \n <ReqBody> END
<ReqOp>  ::= SERVICE | SERVICEN
<ReqBody>::= <Task> \n | <Task> <ReqBody>
<Task>   ::= FUNCTION <FName> \n DATA <Data>
```

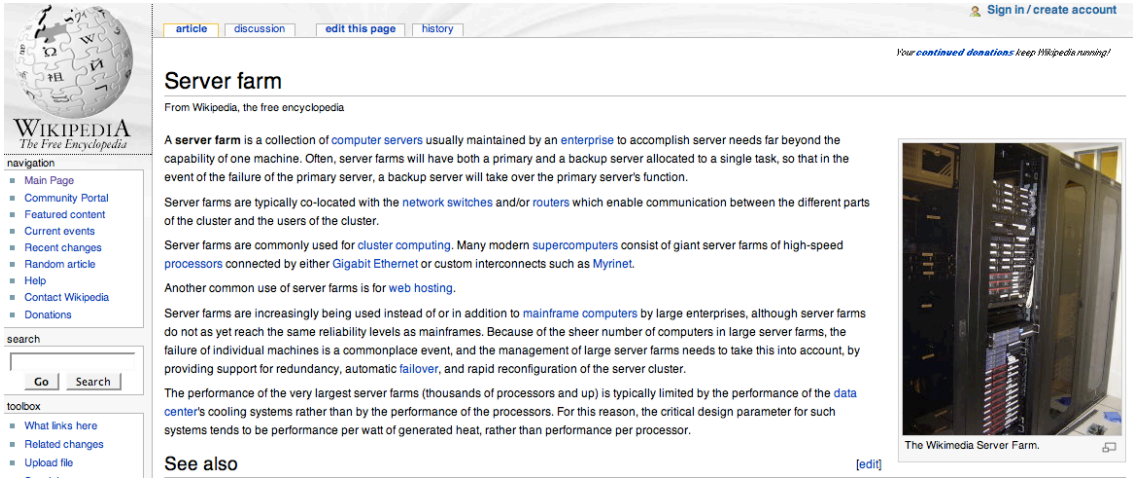

Figure 1: Definizione di server farm da Wikipedia

dove `<FName>` rappresenta un nome di funzione[1] e `<Data>` rappresenta uno o piú dati in formato ASCII[2]. Quindi, ad esempio, messaggi di richiesta validi sono:

```
SERVERFARM SERVICE
FUNCTION Inc
DATA 123
END
```

```
SERVERFARM SERVICEN
FUNCTION Inc
DATA 123
FUNCTION Dec
DATA 321
FUNCTION Mul
DATA 123 321
END
```

```
SERVERFARM SERVICE
FUNCTION Mult
DATA 123 321
END
```

Il server risponderá utilizzando pacchetti nel formato:

```
<AnsMsg> ::= SERVERFARM <AnsType> NUM <Integer>\n <AnsBody> \n END
<AsnType>::= OK | NOK
<AnsBody>::= <Ans> \n | <Ans> <AnsBody>
<Ans>    ::= <Summary> RESULT <Data>
           | <Summary> ERROR <Data>
<Summary>::= ANSWER <FName> <Data> HOST <Hostaddress>
```

(dove `<Integer>` rappresenta un intero e `<HostAddress>` rappresenta un indirizzo IP di host) e quindi messaggi di risposta valida (per esempio in corrispondenza dei messaggi di richiesta visti sopra) saranno del tipo:

```
SERVERFARM OK NUM 1
ANSWER Inc 123 HOST fujih23.cli.di.unipi.it RESULT 124
END
```

oppure

```
SERVERFARM NOK NUM 3
ANSWER Inc 123 HOST fujih23.cli.di.unipi.it RESULT 124
ANSWER Dec 321 HOST fujih12.cli.di.unipi.it ERROR class not found
ANSWER Mul 123 321 HOST fujih8.cli.di.unipi.it RESULT 39483
END
```

Riguardo i messaggi di risposta, la presenza di (almeno) un errore fa' si che il messaggio sia etichettato come `NOK` piuttosto che come `OK`.

Il client puó anche inviare messaggi "di controllo" secondo la seguente grammatica:

[1] nel nostro caso, il nome di un file `.java` che contiene una classe `FName` che implementa l'interfaccia `Compute`

[2] per esempio, la stringa "123 321" oppure la stringa "12.34" sono `Data` validi

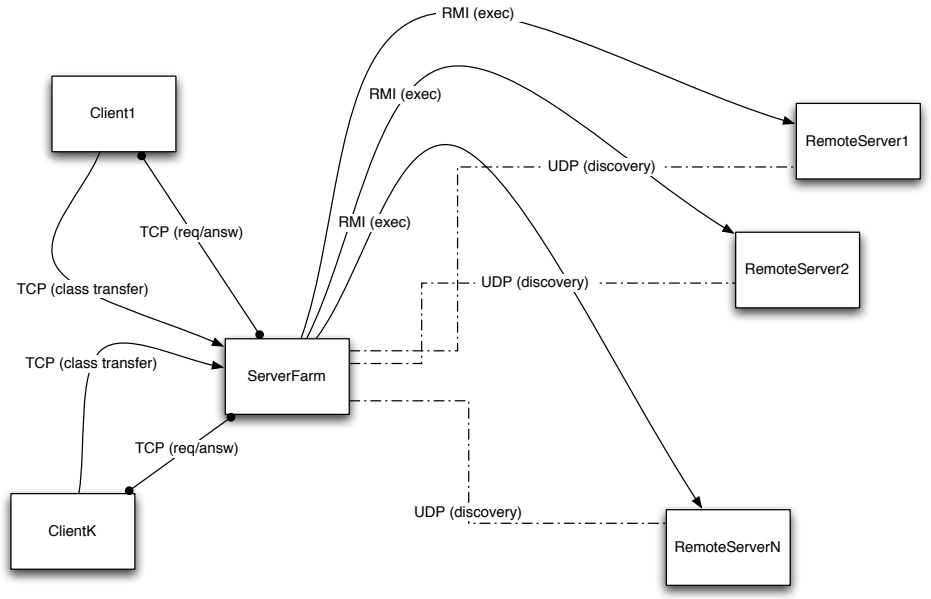

Figure 2: Disegno architetturale del server farm

```
<CtlMsg> ::= SERVERFARM CTL <CtlOp> \n END
<CtlOp>  ::= STATS | SHUTDOWN
```

Come risposta, il server mander un messaggio di OK/NOK secondo la grammatica:

```
<CtlAns> ::= SERVERFARM <CtlATy> \n <CtlBody> \n END
<CtlATy> ::= REPORT | RESET
<CtlBody>::= <Data> \n | <Data> <CtlBody>
```

In particolare, la risposta al messaggio di tipo `REPORT` conterrá una serie di linee che contengono i dati relativi al numero, tipo e indirizzo delle macchine "arruolate" nel server farm nonché il numero di task eseguiti fino a quel momento da ciascuno dei server remoti. La risposta al messaggio di tipo `RESET` sará invece un messaggio di `OK` con le statistiche, ma le statistiche verranno resettate in modo che ognuno dei contatori riparta, ai fini della richiesta successiva, da zero.

3 Disegno architetturale del server farm

Lo schema mediante il quale di deve implementare il server farm é quello della figura 2. Un `ServerFarm` accetta le richieste dai clienti e le passa (per l'esecuzione) ad uno dei `RemoteServer` nel server farm. L'interazione fra cliente e `ServerFarm` avviene mediante socket TCP, quella fra `ServerFarm` e `RemoteServer` avviene mediante UDP (scoperta e arruolamento, raccolta delle statistiche) o RMI (esecuzione remota dei task). In particolare:

- i server interni devono essere realizzati utilizzando RMI. Il processo che interfaccia il server farm con i clienti provvederá a chiamare metodi opportuni dei processi server interni per eseguire i task richiesti
- il processo che interfaccia il server farm con i clienti deve scoprire utilizzando un meccanismo p2p (basato su UDP e multicast) i server interni e mantenerne un catalogo. L'indirizzo di multicast da utilizzare sará `serverfarm.ServerFarm.MULTICASTGROUP`

- il processo che interfaccia il server farm con i clienti deve essere realizzato in modo multithreaded (1 thread, possibilmente da un thread pool, per servire ognuna delle richeste cliente)

4 Codice delle classi che implementano f

Le funzioni f devono essere definite come classi che implementano l'intefaccia `Compute`, cosí definita:

```
public interface Compute {
  public Object exec(Object ... in);
}
```

Ogni funzione nominata nei messaggi di tipo `SERVICE` o `SERVICEN` deve essere presente nella directory in cui viene invocato il cliente. In particolare, per ogni funzione `Fun` nominata in un messaggio di tipo `SERVICE` deve esistere nella directory corrente per il client un file di *Fun.java* o *Fun.class* o *Fun.jar* che contengono il codice che implementa la funzione.

Quando il server riconosce un messaggio di richiesta `SERVICE` con un nome di funzione `Fun`, apre una connessione TCP verso la porta `serverfarm.ServerFarm.CLIENTPORT` del client e si aspetta di ricevere nell'ordine una stringa (caratteri ASCII terminati da un ritorno carrello `\n`) e il contenuto del file che implementa la funzione, sia esso un file `.java`, `.class` o `.jar`, dopodiché utilizza tale file per calcolare il task. In caso di richieste di tipo `SERVICEN`, il server si dovrá aspettare di ricevere una sequenza che contiene: il numero (in caratteri ASCII) delle funzioni, seguito da un ritorno carrello `\n`, tante sequenze nome file funzione, spazio, numero di byte del file (in ASCII) ritorno carrello, contenuto del file, quante sono le funzioni.

5 Clienti

Si devono realizzare diversi clienti, in particolare:

- un `Client1` che prende dalla riga di comando il nome della funziona da calcolare e gli eventuali parametri da passare alla funziona ed invia una singola richiesta al server farm per l'esecuzione di quel task
- un `ClientN` che prende dalla riga di comando il nome di un file che contiene un certo numero di righe, ognuna delle quali e' composta dal nome delle funzione e dai parametri, separati da spazi e manda una richiesta `SERVICEN` al server farm con i task trovati nel file

6 Passaggio dei dati per i task

Si assuma che tutti i dati processati nei vari task siano di tipo intero, e che i vari risultati siano ancora di tipo intero. Di fatto questo significa che il tipo delle funzioni dovrá essere:

```
public interface Compute {
  public int exec(int ... in);
}
```

7 Test del progetto

Il server farm deve girare su macchine Linux configurate come quelle dell'aula H. Verrá testato su macchine del dipartimento, per ragioni di praticitá, ma in caso di malfunzionamenti, contestazioni o altro fará fede l'esecuzione sulle macchine dell'aula H, effettuata in remoto via terminale con ssh.

8 Relazione e modalitá di consegna

La consegna del progetto avviene nelle date stabilite sulla pagina web del corso e consiste in due parti distinte: la consegna, presso il centralino del dipartimento della relazione del progetto (nella cassetta della posta del docente) e la spedizione per email dei sorgenti e del PDF della relazione (all'email del docente).

La relazione deve contenere:

1. una sezione con la descrizione delle scelte implementative non comprese/descritte in questo progetto, con tre sottosezioni, una per il client, una per il server e una per i server remoti
2. una sezione con *tutti* i comandi necessari per far girare il server farm e il cliente, a partire dal file dei sorgenti come consegnato via email
3. un indirizzo di email per le comunicazioni (eventuali) riguardanti lo stato del progetto

La mancanza di una di queste parti, o la sua presentazione in forma incompleta o inutilizzabile o errata comporterá l'impossiblitá di discutere il progetto nella sessione. In particolare, se il progetto non compila o non gira come descritto nella sezione della relazione, verrá respinto e sará necessaria una nuova consegna nella sessione successiva.

Questo progetto vale per tutte le sessioni dell'anno accademico 2006-2007.

9 Svolgimento del progetto

Il progetto va svolto individualmente. Eventuali realizzazioni di gruppo sono accettate purché dichiarate in fase di consegna e purché la discussione avvenga singolarmente.

10 Facoltativo

Alcune delle parti relative a questo progetto potrebbero essere implementate utilizzando tecnologia standard Java, diversa da quella richiesta per lo svolgimento del progetto. Un esempio per tutti é dato dal caricamento dinamico delle classi. Una volta realizzato il progetto come richiesto dalla specifica fornita nelle sezioni precedenti, lo studente puó prendere in considerazione l'implementazione di varianti del progetto che comportino l'utilizzo di strumenti primitivi della piattaforma Java 1.5 per l'implementazione di parti del progetto che nella soluzione originale sono programmati a piú basso livello, secondo le specifiche date. La consegna di eventuali realizzazioni di questo tipo deve comprendere un'opportuna sezione di documentazione delle scelte implementative e degli strumenti utilizzati.