Progetto
Laboratorio Programmazione di rete
Mini-KaZaA

Andrea Di Grazia, Massimiliano Giovine

Anno Accademico 2008 - 2009

Indice

Capitolo 1

Introduzione

1.1 Una panoramica generale.

Il progetto Mini-KaZaA mira allo sviluppo di un sistema p2p per lo scambio di file su WAN, ispirato alla più famosa rete p2p KaZaA. Ogni peer[1] partecipante alla rete Mini-KaZaA condivide un insieme di files con gli altri peer connessi e può ricercare file all'interno della rete e effettuarne il download.

Mini-KaZaA prevede due tipi diversi di peer:

- **Super Nodes (SN):** i SN hanno il compito di gestire le comunicazioni all'interno della rete;
- **Ordinary Nodes (ON):** gli ON hanno responsabilità più limitate, condividono e cercano file nella rete.

Nella rete Mini-KaZaA è prevista anche un'altra entità chiamata **Bootstrap Servers** che contiene la lista di tutti i peer connessi alla rete e dalla quale ogni nodo che desidera entrare a far parte della rete può scaricare la lista aggiornata di tutti i SN presenti.

La rete si costruisce automaticamente dai vari peer secondo un preciso schema e si mantine stabile grazie a processi automatizzati che lavorano in background, completamente trasparenti all'utente.

1.2 La rete Mini-KaZaA.

Ogni peer della rete Mini-KaZaA viene configurato esplicitamente dall'utente al primo avvio come SN o come ON. Successivamente non sarà possibile cambiare tale configurazione.

[1] Ogni nodo della rete è un pari all'interno del network poichè funziona sia da client, per ciò che concerne la ricerca e il download dei file, sia da server per la condivisione dei file o lo smistamento delle ricerche nella rete p2p.

Al momento della connessione alla rete ogni peer, SN o ON, contatta un Bootstrap server che gli fornisce la lista aggiornata di SN presenti in quel momento all'interno della rete.

Un ON sceglie il migliore SN per lui e si connette ad esso. Un SN mantiene in memoria la lista di riferimenti a SN che gli servirà, in un secondo momento, per smistare le interrogazioni. Gli SN, inoltre, avendo un sistema dinamico di connessione ai pari SN esplorano a ogni interrogazione porzioni nuove della rete in modo tale che vi siano il meno possibile porzioni isolate della rete.

Capitolo 2

Bootstrap Server

2.1 Il Bootstrap server in generale

Il Bootstrap server ha il compito di tenere un indice di tutti i SN presenti nella rete che abbiano una certa affidabilità. Per poter fare questo fornisce un servizio di RMI[1] tramite il quale i SN si possono iscrivere alla rete Mini-KaZaA e richiedere liste aggiornate.

Gli aggiornamenti vengono spediti a ogni SN presente nella lista del Bootstrap server tramite un sistema di *callbacks*[2]

Il Bootstrap server deve fornire questo servizio anche agli ON che vogliono entrare nella rete per poter individuare il "miglior"[3] SN al quale potersi connettere. Per questa ragione si è reso necessario indicizzare anche gli ordinary node all'interno del bootstrap server.

2.2 Entriamo nel dettaglio

Il bootstrap server si avvia dal main situato all'interno del file `BootstrapService.java` e subito crea il servizio RMI sulla porta 2008 da mettere a disposizione per i vari nodi della rete con le seguenti istruzioni:

```
1 Registry registry = LocateRegistry.createRegistry(2008);
2 BootStrapServer bss = new BootStrapServer(g,sn_list);
3
4 BootStrapServerInterface stub =
5         (BootStrapServerInterface)
6         UnicastRemoteObject.exportObject(bss, 2008);
```

[1]Remote Method Invocation, tramite questo servizio è possibile invocare metodi che si trovano su una macchina diversa da quella in cui si trova la chiamata a procedura.

[2]Ogni nodo mette a disposizione del Bootstrap server alcune chiamate di procedura che, richiamate, consentono di inviare aggiornamenti.

[3]Spiegeremo nella sezione 4.2 i parametri secondo i quali ogni ON sceglie un SN al quale connettersi.

```
7
8
9  SupernodeCallbacksImpl client_impl =
10         new SupernodeCallbacksImpl(
11         new SupernodeList(),
12         new NodeConfig());
13
14 SupernodeCallbacksInterface client_stub =
15         (SupernodeCallbacksInterface)
16         UnicastRemoteObject.exportObject( client_impl,2008);
```

Con le prime istruzioni il Bootstrap server mette a disposizione tutti i metodi della classe `BootStrapServerInterface` che sono i seguenti:

```
1  public boolean addSuperNode(NodeInfo new_node) throws RemoteException;
2  public boolean removeSuperNode(NodeInfo new_node) throws RemoteException;
3  public boolean addOrdinaryNode(NodeInfo new_node) throws RemoteException;
4  public boolean removeOrdinaryNode(NodeInfo new_node) throws RemoteException;
5  public ArrayList<NodeInfo> getSuperNodeList() throws RemoteException;
```

Con le istruzioni alla riga 9 e 14 registra l'interfaccia di callback `SupernodeCallbacksInterface` che ha i seguenti metodi:

```
1  public void notifyMeAdd(NodeInfo new_node) throws RemoteException;
2  public void notifyMeRemove(NodeInfo new_node) throws RemoteException;
```

Per poter trasmettere e ricevere le informazioni riguardanti i vari nodi della rete, il client Mini-KaZaA e il Bootstrap server usano una classe serializzabile che si chiama `NodeInfo` che analizzamo nella sezione 2.3

2.3 La classe NodeInfo

La classe `NodeInfo` si trova nel package `lpr.minikazaa.bootstrap` ma viene utilizzata da tutto il pacchetto Mini-KaZaA per poter inviare nella rete le informazioni relative ai nodi.

Questa classe rappresenta le informazioni utili di un nodo con le sue variabili private.

```
1  private InetAddress ia_node;
2  private int door;
3  private String id_node;
4  private String username;
5  private SupernodeCallbacksInterface stub;
6  private long ping;
7  private boolean is_sn;
```

Le prime tre variabili private riguardano tutte le informazioni di rete dei nodi, ovvero l'indirizzo IP, la porta di connessione e un id univoco ottenuto facendo la concatenazione della rappresentazione decimale dell'indirizzo IP.

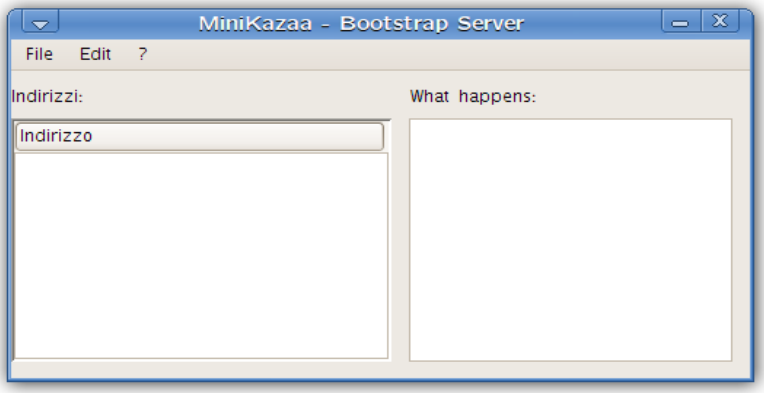

Figura 2.1: Interfaccia grafica del Bootstrap server

La variabile privata `private SupernodeCallbacksInterface stub` rappresenta l'interfaccia per le callbacks che viene messa a disposizione dal nodo. In questo modo il bootstrap server può richiamare direttamente l'interfaccia delle callback di ogni nodo interessato all'aggionrnamento.

L'ultima variabile, `private boolean is_sn` indica se il nodo al quale si riferiscono le informazioni, all'interno della rete ricopre il ruolo di SN o di ON.

La classe contiene tutti i metodi `set` e `get` per poter assegnare valori alle variabili private e per poterne ricavare il contenuto in qualsiasi momento.

2.4 L'interfaccia grafica.

L'interfaccia grafica fornisce le informazioni riguardo a ciò che avviene all'interno della rete.

Uno screenshot dell'interfaccia grafica principale del Bootstrap server si può vedere in Figura 2.1.

Nella parte a sinistra dell'interfaccia vengono inseriti gli *id* di tutti i nodi che si connettono alla rete.

Nella parte destra, invece, vengono visualizzati dei messaggi che spiegano cosa avviene all'iterno della rete.

Capitolo 3

Mini-KaZaA Client

Oltre che di un Bootstrap server, la rete Mini-KaZaA si basa su un client che gli utenti possono usare per accedere alla rete e poter condividere e scaricare file.

3.1 Mini-KaZaA Client in generale

Mini-KaZaA client presenta tutte le funzionalità che consentono una condivisione peer to peer dei contenuti. Ogni client al primo avvio chiede all'utente, tramite un comodo pannello, di scegliere il *ruolo* da interpretare all'interno della rete.

Chi ha più risorse da mettere a disposizione e una banda di comunicazione più ampia può scegliere di essere un Super Node, che oltre a condividere e scaricare, ha la funzione di smistare le query nella rete e accettare richieste direttamente dagli Ordinary Node *figli*. Chi ha meno risorse da mettere a disposizione può scegliere di essere un semplice Ordinary Node.

3.2 Il codice di Mini-KaZaA client

Il codice di Mini-KaZaA client è distribuito in tre diverse librerie:

- **lpr.minikazaa.minikazaaclient**: questa libreria contiene classi comuni a tutti e due i tipi di client dal punto di vista logico. L'esempio più evidente è la classe `MainGui.java`.
- **lpr.minikazaa.minikazaaclient.ordinarynode**: questa libreria contiene le classi che loogicamente appartengono al tipo di client Ordinary Node, ma che, all'occorrenza, possono essere importate anche da un Super Node.
- **lpr.minikazaa.minikazaaclent.supernode**: questa libreria, infine contiene tutte le classi che servono a un supernodo per funzionare

e che appartengono a questo logicamente. Alcune di queste classi, come per esempio `SupernodeCallbacksInterface.java`, vengono utilizzate anche dagli ORdinary Node.

Questa suddivisione è puramente logica visto che i due tipi di client differiscono solo per alcune caratteristiche.

Si è preferito dividere anche le classi che contengono gli stessi task per i SN e per gli ON per poter meglio gestire il codice e renderlo più modulare. Un esempio è rappresentato dalle classi `OrdinarynodeWorkingThread.java` e `SupernodeWorkingThread.java` che hanno lo stesso compito, ma, che piuttosto che complicare con una serie di

```
if <condizione> then
<blocco>
else
<blocco>|
```

si è preferito separare in due classi distinte.

Passiamo ora a una presentazione più particolareggiata del codice comune a Super Node e Ordinary Node.

3.3 Le strutture dati comuni

Per lo sviluppo di Mini-KaZaA è stato necessario predisporre una serie di strutture dati che tutto il software utilizzi per condividere informazioni.

All'interno del package `lpr.minikazaa.minikazaaclient` troviamo le seguenti classi che rappresentano strutture dati comuni a SN e ON:

- `NodeConfig.java`
- `Query.java`
- `Answer.java`
- `SearchField.java`
- `Download.java`
- `DownloadRequest.java`
- `DownloadResponse.java`

Guardiamo cosa si nasconde all'interno di ognuna di queste classi.

3.3.1 NodeConfig.java

La classe `NodeConfig.java` contiene i seguenti attributi:

```
1  private String user_name;
2  private int port;
3  private String bootstrap_address;
4  private int max_conn;
5  private int ttl;
6  private boolean is_sn;
7
8  //Calcolato all'avvio
9  private String my_address;
```

Questi attributi sono i campi che l'utente inserisce nel form al primo avvio del programma e contengono le informazioni di configurazione del nodo.

3.3.2 Query.java

La classe `Query.java` viene utilizzata dal client Mini-KaZaA per l'invio di richieste di file nella rete.

Contiene diversi attributi per i quali ci sono i metodi `set` e `get`. Questa classe inoltre implementa le interfacce `Serializable` e `Cloneable`. La prima serve per poter inviare su rete come flusso di byte l'oggetto `Query`. La seconda invece serve per poter copiare un'istanza dell'oggetto `Query` in una seconda istanza.

```
1  private String body_q;                  //Regex of a sended query
2  private Answer body_a;                  //Answer query
3  private OrdinarynodeFiles body_f;       //Notify a supernode to
4                                              //have many files to share.
5  private NodeInfo id_origin;             //Source of a query
6  private NodeInfo sender;                //NodeInfo of sender node.
7  private NodeInfo receiver;              //NodeInfo of receiver node
8
9  private int ttl;                        //Time to live of a query
10 private int id;                         //Id of origin query
```

La classe `Query` ha tre gruppi di attributi.Un primo gruppo descrive il contenuto della query e di conseguenza il tipo di query. Un secondo gruppo serve per identificare i soggetti coinvolti nello scambio della query stessa. Il terzo gruppo contiene invece parametri per l'identificazione della query.

Analizziamo uno ad uno questi parametri per capire meglio come funzionano le query in Mini-KaZaA.

- `body_q`: il vero corpo della query di richiesta di un file. È una stringa che contiene un espressione regolare che il client Mini-KaZaA riesce a interpretare;

- `body_a`: la parte dell'oggetto `Query` che contiene la risposta a una determinata richiesta. Analizzaremo la classe `Answer` successivamente;
- `body_f`: questo campo viene riempito da un ON che vuole inviare al proprio SN la sua lista di file per metterli a disposizione di tutti;
- `id_origin`: per ogni query deve essere nota l'origine dalla quale proviene la query stessa per poi poterla correttamente fermare al punto giusto e farla ritornare al mittente. Questo è il compito del campo `id_origin`;
- `sender`: questo campo indica uno dei due soggetti che sono impegnati in un singolo scambio di query, il nodo da cui parte;
- `receiver`: questo campo indica il nodo a cui deve arrivare la query in uno scambio;
- `ttl`: questo campo sta per *Time To Live* e indica il numero di scambi per il quale la query deve continuare a esistere. Serve principalmente per evitare che si creino dei cicli infiniti di scambio della query ottenendo quindi una valanga di dati ridondanti con conseguente intasamento della rete;
- `id`: ogni nodo può inviare più query alla volta nella rete e il compito di questo campo è di identificare univocamente la query presso il suo nodo origine.

3.3.3 Answer.java

La classe `Answer.java` contiene i file che possono corrispondere ai criteri di una ricerca.

È una classe molto semplice ma molto utile per indicizzare rapidamente i file.

Ecco il codice nel quale vengono dichiarati gli attributi della classe.

```
1 //List of files that corresponding to a query.
2 private ArrayList <OrdinarynodeFiles> files;
3
4 //Id of origin query
5 private int id;
```

L'attributo `files` è una lista di OrdinarynodeFiles, Sezione 4.1.1.

La classe `Answer.java` viene utilizzata

L'attributo `id` richiama semplicemente l'id univoco della query di cui fa parte l'oggetto `Answer`.

3.3.4 SearchField.java

La classe `SearchField.java` viene utilizzata dal client Mini-KaZaA per tenere in memoria tutti i risultati associati a una richesta di file. Da uno di questi campi poi vengono estratte le informazioni per eventuali download di file.

Anche questa classe è piuttosto semplice poichè funziona da appoggio alla rappresentazione grafica e per snellire la quantità di informazioni da tenere in memoria per l'utente.

Il codice che descrive gli attributi della classe è il seguente:

```
1 //File owner
2 private NodeInfo owner;
3
4 //File descriptor
5 private MKFileDescriptor file;
```

Con queste due semplici informazioni è possibile sia risalire al proprietario, compreso l'indirizzo ip da contattare per il download, sia ottenere tutti i metadati del file da scaricare[1].

3.3.5 Download.java

3.3.6 DownloadRequest.java

3.3.7 DownloadResponse.java

3.4 Il percorso di una query

3.5 La grafica del client Mini-KaZaA

[1] I download così come le ricerche vengono effettuati mediante l'hash univoco md5 che tratteremo nella Sezione 6.1

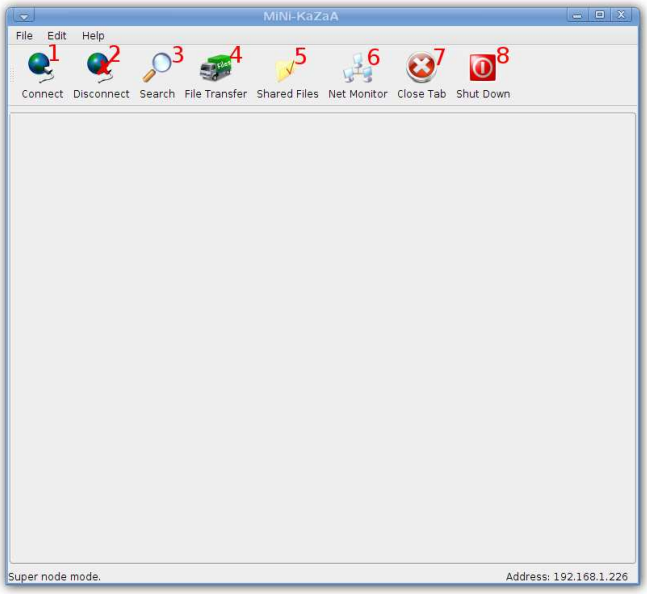

Figura 3.1: L'interfaccia grafica principale del client.

Capitolo 4

Ordinary Node

4.1 Le classi del package lpr.minikazaaclient.ordinarynode

4.1.1 OrdinarynodeFiles.java

4.2 Scelta del SN al quale connettersi

Capitolo 5

Super Node

5.1 L'interfaccia per le callback

Capitolo 6

Il package Util

6.1 Calcolo dell'md5

Capitolo 7

Il package di grafica

Appendice A

Manuale d'uso