Jolella – Condivisione Files P2P

Laboratorio Sistemi Operativi A.A. 2018-2019

Nome Cognome, Nome Cognome, Nome Cognome

A.A. 2018-2019

June 27, 2019

Abstract

Il progetto propone la creazione un sistema distribuito per la condivisione di file decentralizzato, in una rete di entit pari tra loro. La realizzazione del progetto segue i principi della programmazione orientata ai servizi, rispettandone l'architettura e i paradigmi di gestione della concorrenza. I servizi e la comunicazione tra di essi sono implementati nel linguaggio di programmazione Jolie.

Indice

1 Descrizione del progetto

something something jolie

2 Contatti

Contatti del gruppo

- Nome Cognome[a], Matr. 000099999,
  nome.cognome@studio.unibo.it
  mail@gmail.com
- Nome Cognome, Matr. 000099999,
  nome.cognome@studio.unibo.it
  mail@gmail.com
- Nome Cognome, Matr. 000099999,
  nome.cognome@studio.unibo.it
  mail@gmail.com
- Nome Cognome, Matr. 000099999,
  nome.cognome@studio.unibo.it
  mail@gmail.com

[a]Referente del gruppo

3 Istruzioni

Le istruzioni per l'esecuzione della demo sono nel seguente ordine:

1. avvio del logger;

```
jolie logger.ol
```

2. avvio del primo jeer;

```
jolie -C "location=\"socket://localhost:9001\"" jeer.ol
```

3. avvio del secondo jeer;

```
jolie -C "location=\"socket://localhost:9002\"" jeer.ol
```

4. avvio del terzo jeer;

```
jolie -C "location=\"socket://localhost:9003\"" jeer.ol
```

5. avvio del quarto jeer;

```
jolie -C "location=\"socket://localhost:9004\"" jeer.ol
```

6. avvio del file per l'esecuzione della demo.ol

```
jolie demo.ol
```

La selezione delle porte non vicolante in senso vero e proprio, possibile infatti creare un numero indeterminato di jeer che si collegano alla rete con una porta arbitraria. Le porte indicate tuttavia sono necessarie ai fini dell'esecuzione della demo.

4 Strategie di implementazione

In questa sezione descriveremo le scelte implementative del progetto.

4.1 Struttura

4.1.1 Struttura della demo

```
Logger@LoggerPort("Hi my name is Demo1 and I am ready")(loggerResponse);
println@Console( "Starting DEMO1" )();
demo1.filename="demo1.txt";
demo1.destination="socket://localhost:9003";
OutPort.location="socket://localhost:9001";
Demo@OutPort(demo1)(response);
if (response){
  println@Console( "demo1 executed" )()
};
Logger@LoggerPort("Hi my name is Demo1 and I finished")(loggerResponse);
```

Il servizio di Demo si occupa di comunicare ai vari jeer le istruzioni da eseguire. Questo servizio quindi una demo non interattiva poich va a sostituire gli input dell'utente e risulta molto comodo sia per fini dimostrativi che di testing. I dati che vengono comunicati al jeer indicato dalla location dell'OutputPort sono il nome del file da richiedere e il nodo da cui lo si andra a scaricare. E' necessario evidenziare che nonostante venga comuniato il nodo da cui scaricare il file, ci non preclude l'esecuzione della ricerca del file all'interno della rete. Questo dato, infatti, stato introdotto per evitare situazioni di ambiguit quando pi nodi possiedono lo stesso file, rendondo necessaria una selezione del jeer dal quale scaricare il file.

4.1.2 Struttura del logger

```
execution {concurrent}
 init {
  global.counter=0
}

main{
 [Logger(MessageData)(LoggerResponse){
  synchronized( token ){
   println@Console(global.counter+". "+MessageData)();
   global.counter++;
   LoggerResponse=true
  }
 }]
}
```

Il logger offre un servizio che si occupa di mostraze in un'unica console i messaggi ricevuti dai componenti della rete in modo ordinato e numerato. Il servizio messo a disposizione molto semplice e consiste nello stampare ogni messaggio ricevutto preceduto da un contatore. Il contatore dichiarato come variabile globale in modo da tracciare la numerazione nelle varie istanze del programma. Per gestire la concorrenza abbiamo scelto di usare il costrutto synchronized che permette un singolo accesso alla porzione di codice sensibile al problema di concorrenza.

4.1.3 Struttura di un jeer

Ogni singolo jeer strutturato in modo da essere modulare. Per prima cosa andiamo ad evidenziare alcune caratteristiche di un jeer.

```
execution {concurrent}
```

L'utilizzo di execution {concurrent} permette al jeer di eseguire pi istanze ed operare in parallelo, favorendo la concorrenza delle operazioni.

```
init {
  println@Console( "Start node init" )();
  install(TypeMismatch => println @Console("TypeMismatch: " + main.TypeMismatch)()) ;
  global.status.myLocation=location;
  global.jeertable.node[0].location = global.status.myLocation ;
  Logger@LoggerPort("Hi my name is "+location+" and I am almost ready")(loggerResponse);
  println@Console(loggerResponse)();
  SplitRequest=location;
  SplitRequest.regex=":";
  split@StringUtils(SplitRequest)(SplitResponse);
  global.porta=SplitResponse.result[#SplitResponse.result-1];
  exists@File(global.porta)(existResponse);
  println@Console("existResponse: "+existResponse )();
  if(!existResponse)
  {
    mkdir@File(global.porta)()
  }else{
    ListRequest.directory=global.porta;
    list@File(ListRequest)(ListResponse);
    for ( i=0, i<#ListResponse.result, i++ ) {
    global.jeertable.node[0].filelist[#global.jeertable.node[0].filelist]=ListResponse.result[i]
    }
  };
  if (global.status.myLocation == RootLocation+"1") {
   println@Console( "I'm a lonely jeer :(" )()
  } else {
   findJeer@findInternalJeer(global.jeertable)(findJeerResponse);
   for ( i=0, i<#findJeerResponse.node, i++ ) {
     global.jeertable.node[#global.jeertable.node]=findJeerResponse.node[i]
   }
 };
  Logger@LoggerPort("Hi my name is "+global.status.myLocation+" and I'm definitly ready!")(loggerRespo
println@Console( "Node initialization finished" )()
}
```

Durante l'inizializzazione del jeer vengono create alcune variabili globali, accessibili a tutte le istanze del programma, che sono utili per accedere ad alcuni dati spesso usati dei servizi che successivamente saranno implementati. Inoltre, si controlla che esista una cartella relativa al jeer: nel caso non esista viene creata, se invice esiste si contolla quali file siano presenti. Successivamente si esegue una ricerca degli altri jeer all'interno della rete.

```
[Demo(Value)(DemoResponse) { ...
...
```

.. e qua basta descrivere tutte le input choice

```
embedded service 1
...
```

.. e qua basta descrivere i servizi embeddati

4.2 Features

In questa sezione discuteremo alcune delle features pi rilevanti che sono state implementate nel progetto.

4.2.1 Decentralizzazione

//un pezzo di codice

Una sezione fondamentale del progetto la gestione della rete. A fronte delle varie possibilit presenti per modellare la rete abbiamo optato per una rete completamente decentralizzata. Similarmente a gnutella, il funzionamento della rete non dipende da alcun server centrale. Il punto pi critico per costruire una rete peer-to-peer decentralizzata consiste nel setup iniziale durante il quale un nodo, nel nostro caso il jeer, deve fare bootstrap e trovare almeno un altro nodo. Nel protocollo Gnutella storicamente sono stati usati vari metodi per fare il bootstapping di un nodo, tra i quali includere nell'applicativo una lista pre-esistente di nodi. Essendo questo l'unico metodo implementabile semplicemente in Jolie abbiamo deciso di adottarlo. Una volta connesso ad un nodo pre-esistente, il nodo chiede una lista di nodi attivi ai quali si connetter. Ci in cui la nostra implementazione della rete P2P si differenzia da Gnutella la gerarchia della rete. In Gnutella infatti sono presenti leaf nodes e ultra nodes in cui i leaf nodes si connettono al pi a 3 ultrapeer e gli ultrapeer si connettono ad altri 32 ultrapeer. Nella nostra implementazione abbiamo ritenuto che questa distinzione non fosse ne significativa ne vantaggiosa. Aprofittando infatti della necessit di avere una fase di bootstrapping possibile connettere ogni jeer alla rete in modo paritario. Questo aproccio astuto semplifica notevolmente la gestione della rete e rende marginale il meccanismo di flooding, riducendolo ad un broadcast. Sebbene questa soluzione non sia al momento scalbile poich introduce un notevole overhead nelle comunicazioni, ci ci permette in una rete di piccola/media dimensione di costruire una rete altamente ridondante e robusta. Inoltre, possibile ovviare a questo svantaggio aggiungendo dei semplici meccanismi di controllo nella gestione delle connessioni, mantenendo i vantaggi di una rete completamente connessa.

4.2.2 Trasferimento file

//un pezzo di codice

Realizzare il trasferimento di file ci ha posto davanti ad una scelta tra due alternative, entrambe con vantaggi e svantaggi. Il primo modo che ci venuto in mente, che anche quello relativamente pi semplice, consiste nel copiare il file dalla sua cartella di origine alla cartella del jeer che lo ha richiesto, facendo eseguire il totto dal jeer che possiede fisicamente il file. In alternativa possibile trasferire i dati del file al jeer he lo ha richiesto e delegare ad esso il compito di creare il file e popolarlo con i dati. Dopo un'attenta analisi abbiamo optato per quest'ultima metodologia in quanto risulta pi in linea con un'applicazione client/server che non venga eseguita solo in localhost ma anche attraverso la

rete. Urilizzare i servizi readFile e WriteFile permette di gestire il contenuto di un file indipendentemente dalle caratteristiche del contenuto, pi precisamente indipendentemente dal formato dei dati.

4.2.3 Ricerca file

```
//un pezzo di codice
```

La ricerca di un file all'interno della reste stata implementata utilizzando una sorta di cache locale. Durande la comunicazione con nuovi nodi oltre all'elenco dei Jeer tramite le rispettive location anche l'elenco dei file per ogni jeer. Questo permette di avere una cache aggiornata di frequente, rendendo la ricerca di un file pi veloce. Nel caso in cui il file non venga trovato viene eseguito un aggiornamento della lista, permettendo di controllare nuovamente la disponibilit del file. La scelta di includere la lista dei file nelle richieste di discovery di nuovi jeer permette ad un jeer appena creato di essere immediatamente pronto per la ricerca di un file, una situazione ottimale se si ipotizza un caso d'uso in cui un utente scarica il programma e vuole avere un'overview dei file disponibile nella rete.

4.2.4 Rimozione jeer

```
//un pezzo di codice
```

Pre rendere possibile ad un jeer nella rete di essere correttamente rimosso abbiamo scelto di implementare un servizio dedicato che invia un messaggio di rimozione a tutti i nodi presenti nella lista dei jeer. Ci che viene trasmesso la location che caratterizza il nodo da rimuovere. Il meccanismo di comunicazione del tutto analogo alla ricerca di altri jeer, infatti sarebbe stato possible gestire queste due features attraverso lo stesso servizio trasmettendo diversi dati. Abbiamo scelto di separarli per evitare di dover aggiungere una serie di controlli sul messaggio che avrebbero reso il codice poco mantenibile, violando inlotre il principio di single responsibility.

4.2.5 Gestione connessioni

Per quanto riguarda la gestione del numero massimo di connessioni possibile gestire questo aspetto tramite un contatore dichiarato attraverso una variabile globale. Abbiamo evitato di implementare questo meccanismo per i pochi vantaggi che presenterebbe. Non inserire questo contatore infatti permette di mantenere la sezione di codice lock-free. Altre ragioni che hanno supportato la nostra scelta sono la volont di delegare il controllo della congestione della rete al protocollo TCP nel transport-layer e la permaneza dell'overhed della connessione se gestita tramite un contatore nell'application layer. Queste ragioni sono strettamente correlate: gestire questo aspetto ad alto livello non porta a vantaggi comparabili con la gestione a livello di sessione. In Jolie presente la primitiva "spawn" che permetterebbe di astrarre la gestione della sessione permettendo di usare il dynamic parallelism per istanziare un numero predeterminato di istanze. Nonostante questa primitiva risulti molto appetibile per essere applicata a questo caso, le limitazioni che essa presenta, poich le sessioni

istanziate non possono contenere operazioni di input, avrebbere reso necessario un'inversione delle comunicazioni, implementando un modello di comunicazione asincrona e la gestione della correlazione tra le sessioni.