Laboratorio di Programmazione di Rete-B
Laurea Triennale in Informatica. a.a.08/09
Progetto di fine Corso

BitCreek: Una Rete P2P per la Distribuzione di Contenuti

Versione 0.8 – Andrea Corradini, Laura Ricci, Daniele Sgandurra

1 Scopo del Progetto

Il progetto richiede la realizzazione di *BitCreek*, una *Content Distribution Network, (CDN)* ispirata a *BitTorrent* [1, 2].
In *BitTorrent*, un peer, il *Seeder*, pubblica un file F mediante la creazione e la pubblicazione di un *descrittore* di F (il file *.torrent*) su un insieme di *Server* di indirizzo noto. Il file viene decomposto in *pezzi* della stessa dimensione e, per ogni pezzo, viene calcolato l'hash mediante *SHA1*. Ogni pezzo viene quindi decomposto in *blocchi* che rappresentano l'unità base di trasferimento dell'informazione sulla rete. Il descrittore del file contiene il nome del file, la sua lunghezza, la sequenza di valori calcolati mediante *SHA1* ed altre informazioni.
Al momento della pubblicazione del descrittore viene attivato anche un *tracker*, ovvero un servizio che *coordina la distribuzione* del file ai peer che lo richiedono. Un riferimento al tracker è memorizzato all'interno del descrittore del file.
Un peer che intende scaricare F, deve prima reperire il suo descrittore, e quindi contattare il relativo *tracker* per notificargli la propria identità ed ottenere da esso la lista degli altri peer che stanno partecipando alla distribuzione di F. L'insieme dei peer che partecipano alla distribuzione di uno stesso file viene indicato in *BitTorrent* con il termine *swarm*. Si noti che ad uno swarm partecipano sia peer *Seeder* che hanno pubblicato/scaricato l'intero file che peer *Leacher* che ne posseggono alcune parti e devono terminare il download di altre parti.
Dopo aver ottenuto dal tracker la lista dei peer partecipanti allo swarm, ogni peer tenta di stabilire una connessione con ciascuno di essi. Alcuni tentativi di connessione possono fallire perchè ogni peer può gestire un numero limitato di connessioni. Per ogni connessione stabilita, viene eseguito un *handshake* in cui i peer si scambiano una *lista con i riferimenti ai pezzi posseduti*. In questo modo un peer possiede la conoscenza della distribuzione dei pezzi su un sottoinsieme dei peer dello swarm e può quindi selezionare i pezzi da scaricare ed il/i peer da cui conviene effettuare il download. Dopo aver scaricato completamente un pezzo, un peer ne verifica l'integrità calcolando l'SHA1 del pezzo e confrontandolo con quello contenuto nel descrittore del file. Successivamente, il peer notifica agli altri peer con cui è connesso l'acquisizione del pezzo. Notare che un peer che rimane all'interno dello swarm dopo il download dell'intero file diventa un *Seeder* e contribuisce alla distribuzione

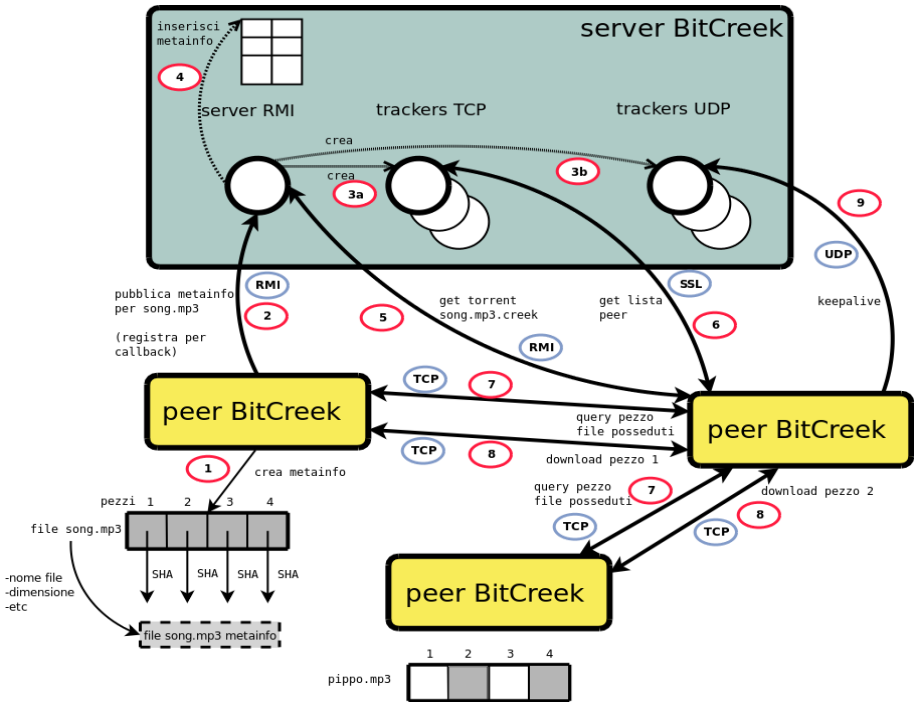

Figure 1: BitCreek: l'Architettura

del file. Si noti anche che se il *Seeder* iniziale lascia la rete prima che ogni pezzo del file sia stato replicato su almeno qualche peer dello swarm, nessun peer è in grado di ricostruire l'intero file.
BitTorrent definisce un protocollo per determinare l'ordine con cui devono essere scaricati i pezzi che compongono un file e le modalità con cui vengono scelti i peer da cui scaricare tali pezzi.
Il prossimo paragrafo illustra il protocollo di *BitCreek*, che si ispira a *BitTorrent* e costituisce la specifica del progetto da realizzare per LPR.

2 BitCreek: Specifica ed Implementazione

In *BitCreek* esiste, per semplicità, un singolo server chiamato *BitCreekServer*, che gestisce i descrittori dei file pubblicati. Ogni peer, chiamato *BitCreekPeer*, conosce l'indirizzo del *BitCreekServer*.
L'architettura generale di *BitCreek* è mostrata in Fig.1. Nella figura viene mostrata la struttura dei *BitCreekPeer* e del *BitCreekServer* e le interazioni tra le componenti dell'applicazione.
Consideriamo la pubblicazione di un file F. Il peer che pubblica F, crea il descrittore di F *(1)*, quindi lo pubblica sul *BitCreekServer*. Il descrittore deve contenere almeno

il nome di F, la sua lunghezza, e una stringa ottenuta concatenando l'*SHA1* calcolato su ogni pezzo di F. Per la pubblicazione *(2)* è necessario utilizzare il protocollo *RMI*. Contestualmente alla registrazione del descrittore, un peer registra anche una *callback* affichè gli venga in seguito notificata l'identità di ogni altro peer che effettua una ricerca del descrittore di F. Quando il *BitCreekServer* riceve il descrittore di F, ci aggiunge i numeri di porta dove sono attivi due thread: il *Tracker TCP*, *(3a)* e il *Tracker UDP*, *(3b)*. Il primo gestisce lo swarm associato ad F, mentre il secondo si occupa di ricevere i messaggi di keep-alive dai peer appartenenti allo swarm. Il descrittore viene quindi registrato in una struttura dati *(4)* del *BitCreekServer*, mentre al peer che ha pubblicato il descrittore di F vengono inviati (sfruttando la callback) le porte dei due *Tracker*.
Supponiamo ora che un *BitCreekPeer* intenda scaricare F. L'interazione con il *BitCreekServer* per la ricerca del descrittore di F *(5)* avviene tramite RMI; quindi dal descrittore si ricava il riferimento alla porta su cui è in ascolto il relativo *Tracker TCP* e lo si contatta mediante *SSL (6)* per ottenere la lista dei peer. Le connessioni dirette tra i peer dello swarm devono essere implementate usando il protocollo *TCP*. Su queste connessioni vengono inviate sia le richieste per le liste dei pezzi *(7)* che il download dei pezzi *(8)*. Infine ogni peer utilizza il protocollo *UDP* per inviare periodicamente messaggi di *keep-alive (9)* al *tracker UDP* per notificare la propria permanenza sulla rete.

Il progetto deve soddisfare inoltre i seguenti requisiti:

1. La dimensione dei pezzi dei file deve essere di $4Kbyte$: un pezzo rappresenta l'unità base di trasferimento.

2. Ogni *Tracker TCP* può gestire al massimo H descrittori di file, dove H è una costante configurabile a tempo di compilazione. Il *BitCreekServer* attiva un nuovo *Tracker TCP* quando tutti quelli attivi hanno già H descrittori di file da gestire.

3. Il *Tracker TCP* che gestisce il descrittore di un file F deve mantenere aggiornata la lista dei peer appartenenti allo swarm di F, inserendo i nuovi peer interessati a scaricare F e cancellando coloro che non mandano un messaggio di *keep alive* da un certo tempo T, configurabile a tempo di compilazione. Descrivere esplicitamente quando uno swarm viene considerato vuoto e garantire che il descrittore del file venga eliminato.

4. Le interazioni tra il *BitCreekServer*, i *Tracker TCP*, e il *Tracker UDP*, che sono tutti thread sullo stesso host, devono avvenire tramite strutture dati condivise.

5. Ogni *BitCreekPeer* legge da un file di testo le azioni che deve eseguire, e scrive su di un file di testo il log delle operazioni effettuate. Per le azioni da eseguire,

decidere una opportuna sintassi e documentarla nella relazione. Devono essere previsti almeno i seguenti tipi di azione: (1) pubblicazione di un file; (2) query e conseguente download di un file; (3) uscita dallo swarm di un file; (4) wait per X millisecondi.

Nel file di log vanno descritte, una per riga, tutte le operazioni salienti (e il relativo esito) dei processi di pubblicazione, di query e di download di file, compreso le connessioni stabilite con i peer.

6. Ogni peer può supportare al massimo K connessioni. Inoltre ogni peer può stabilire al massimo N connessioni al momento della sua unione allo swarm, con $N < K$ mentre le restanti $K - N$ connessioni vengono utilizzate per accettare connessioni da peer che si uniscono successivamente allo swarm. K e N devono essere configurabili staticamente. Il peer deve usare un Thread Pool Executor per gestire le connessioni.

7. Definire e documentare con chiarezza una opportuna strategia con cui un peer seleziona, dopo l'entrata in uno swarm, i pezzi da scaricare ed i peer da cui devono essere scaricati tali pezzi.

Tenere presente anche i seguenti consigli:

- Per la strategia del punto 7, si può prendere spunto dalle politiche definite da *BitTorrent*, ad esempio:
  - *rarest pieces first*: i pezzi con il minor numero di repliche all'interno dello swarm vengono scaricati per primi. Questa strategia tende a replicare i prezzi più rari, favorendone la disponibilità.
  - *endgame*: quando il download del file è prossimo al termine, i pezzi che servono per completare il download del file vengono richiesti a tutti i peer con cui si è connessi. Questa strategia evita che il downlaod di un singolo pezzo ritardi il completamento dell'intero download.
  - per quanto riguarda la scelta dei peer da cui scaricare i pezzi, si può utilizzare una semplice scelta casuale di un peer tra tutti quelli che possiedono il pezzo.

- Per calcolare l'SHA1 di ogni pezzo del file, è possibile utilizzare la libreria $JAVA$ reperibile al seguente indirizzo:

  http://java.sun.com/javase/6/docs/api/java/security/MessageDigest.html

L'implementazione di ogni ulteriore funzionalità del protocollo *BitTorrent* inciderà positivamente sulla valutazione del progetto. A tale proposito è possibile consultare

la specifica del protocollo definita in [1, 2]. Ad esempio è possibile definire strategie di selezione dei pezzi e dei peer più sofisticate. In *BitTorrent* ogni peer sceglie periodicamente un numero limitato di peer dello swarm verso cui intende effettuare l'upload ed effettua l'*unchocking* delle connessioni verso questi peer, mentre le comunicazioni con i restanti peer subiscono un *chocking* temporaneo, cioè viene interrotto temporaneamente l'upload verso questi peer. I peer *unchoked* vengono scelti in modo da selezionare quelli che hanno offerto una miglior velocità di download nell'ultimo periodo di tempo. Inoltre, un peer scelto casualmente viene sempre inserito tra i peer *unchoked*. Questo consente di valutare la velocità di download da nuovi peer e di consentire il bootstrap di nuovi peer sulla rete.

3 Modalità di svolgimento del Progetto

Il progetto può essere svolto in gruppo. Ogni gruppo deve essere composto al massimo da *due studenti*. Il materiale consegnato deve comprendere:

- La stampa di tutto il codice dello strumento e di eventuali programmi utilizzati per il test delle funzionalità dello strumento.
- Una stampa della relazione *in formato pdf* che descriva tutte le scelte effettuate. La relazione deve contenere
  - una descrizione generale dell'architettura del sistema e della scelte di progetto effettuate.
  - una descrizione delle strutture dati utilizzate
  - uno schema generale dei threads attivati da ogni *BitCreekPeer* e dal *BitCreekServer* e delle modalità di interazione di tali threads
  - un manuale d'uso che indichi chiaramente le modalità di interazione con il *BitCreekPeer* per monitorarne le funzionalità.

  L'organizzazione e la chiarezza dell'esposizione della relazione influiranno sul voto finale dell'esame. L'utilizzo di metodologie di documentazione del software quali diagrammi UML (delle classi, di sequenza,...) sarà considerato positivamente ai fini della valutazionde del progetto.

Consegnare Relazione e codice sia in formato cartaceo, presso la portineria del Dipartimento, sia in formato elettronico, via e-mail.
Il progetto deve essere consegnato una settimana prima della data dell'orale. L'orale verterà sia sulla discussione del progetto che sul programma svolto durante il corso.
Il progetto può essere realizzato su qualsiasi piattaforma di sviluppo e sistema operativo, ma è necessario che sia compilabile ed eseguibile sulle macchine Linux presenti

nel Polo Didattico (in particolare gli host `reckonXX` del Laboratorio 1, e sugli host `fujiXXX` dei Laboratori H e M), versione 1.6 di Java. Prima di consegnare il progetto verificare la compilazione e l'esecuzione dell'applicazione su tali host (anche tramite `ssh`), utilizzando host diversi (evitare l'uso di *localhost*). Dato che al Polo l'home directory di ogni utente è esportata tramite NFS, eseguendo il progetto su host diversi, le directory saranno condivise. Per evitare problemi su file condivisi dai peer del progetto (esempio file di configurazione, directory di download/upload) ci sono due possibilità:

- ogni peer deve essere eseguito in una directory distinta dalle altre;
- il nome dei file di configurazione/download condivisi devono essere resi unici per ogni peer (ad es., usando l'IP all'interno di tali nomi).

Al momento della consegna inviare un file compresso in formato `tgz` o `zip` (non `rar`) contenente i sorgenti java, eventuali file jar con librerie di supporto, file di configurazione e immagini se presenti. Non inserire i file class nel file compresso. Inserire nella directory radice del progetto un file di testo chiamato `README` in cui deve essere specificato il/i comando/i per compilare il progetto da riga di comando (tramite `javac`) su Linux. Specificare, se sono presenti, anche le opzioni di compilazione (ad esempio `-classpath` etc).

Attenzione: il progetto deve compilare da riga di comando senza errori e senza warning sugli host `reckonXX` del Laboratorio 1, e sugli host `fujiXXX dei Laboratori H e M`. Specificare nel `README` come avviare da riga di comando (tramite `java`) le singole componenti (anche in questo caso, se presenti, specificare i parametri di avvio), e altre informazioni che possono essere utili ai docenti del corso per il test dell'applicazione. Al momento della verifica del progetto verrà effettuato dai docenti un "copia ed incolla" dei comandi presenti nel file `README` per compilare ed eseguire le applicazioni da riga di comando. Per questo motivo, verificare attentamente che eseguendo i comandi presenti nel `README` la compilazione e l'esecuzione dell'applicazione sia corretta e non richieda la modifica dei sorgenti (sono accettabili richieste di modifiche a eventuali file di configurazione e/o la specifica di parametri da riga di comando, che comunque devono essere dettagliatamente specificate nel `README`).

References

[1] T.V.L.Michelle
A BitTorrent Implementation and Simulation,
Project Report, NationalUniversity of Singapore.

[2] *BitTorrent Specification*
http://wiki.theory.org/BitTorrentSpecification