UNIVERSITÀ DI PISA

Master of Science in Computer Engineering

Project Report

WebTorrent

Distributed Systems and Middleware Technologies

T. Billi, E. Casapieri, A. Di Donato

Academic Year 2020/2021

Contents

Chapter 1

Introduction

1.1 Overall idea

As a project, we have designed and developed our own free interpretation of the famous BitTorrent protocol. In particular, a user interested in downloading/uploading a file must necessarily have an account within the WebTorrent platform. Once the access credentials have been obtained, following a signup procedure, and after indicating the address and port at which he can be reached by other peers, he is able to freely upload or download a file from the WebTorrent network. To be able to download a file, the first thing to do is to find it in the appropriate section of the platform; once this has been done the user will obtain a configuration file, which after being inserted into the BitTorrent client, will allow the exchange of chunks of the desired file with the other peers. If instead the user decides to upload a file, in the appropriate section, this will be registered within the tracker responsible for the management of that file together with the other owners of at least one chunk of that file, if any; in this way subsequent requests for the file just uploaded will probably be satisfied even by conctacting the peer that has made itself available for the transfer of the file.

1.2 Assumptions

The following are the assumptions made during the development of the project:

- The list of peers to exchange chunks with is a random subset of all peers with at least one chunk of the requested file.
- No chocking peers policy has been implemented
- Only one file can be downloaded per torrent client instance at a time
- For a given file, we have only one tracker responsible for managing peers with at least one of its chucks.
- We were forced to use an instance of the Mongo database locally due to a limitation in the use of the jdk version imposed by glassfish5 requirements. This version did not support the libraries needed to connect to remote mongo instances.

Chapter 2

Application Infrastructure

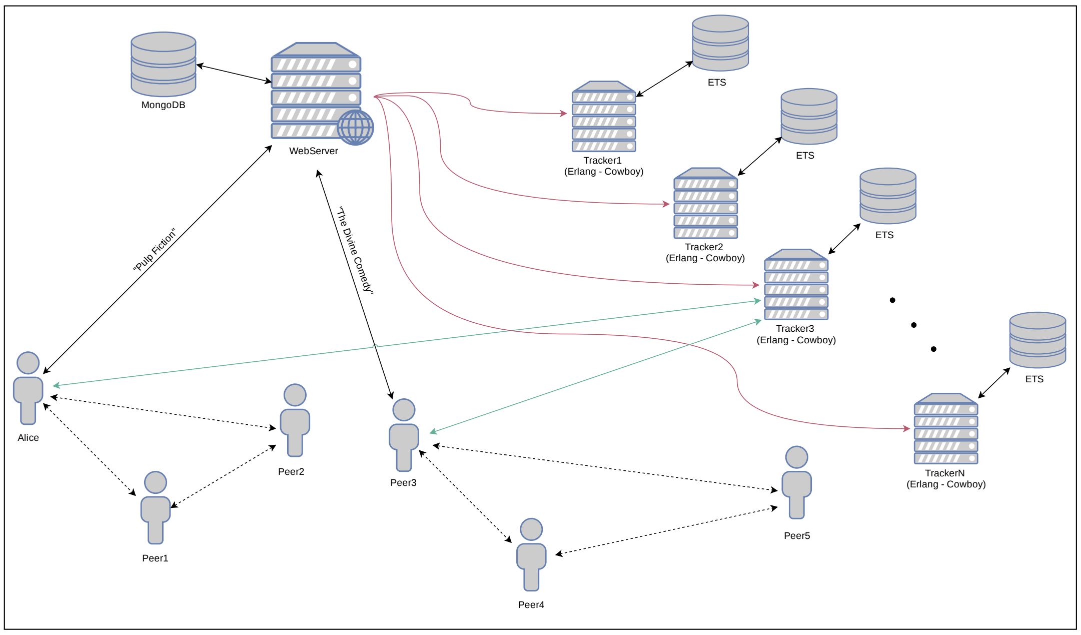

Figure 2.1: WebTorrent infrastructure

The main components of the infrastructure with the main tasks performed, as well as the technologies that were used are:

- **Application Server**: It is in charge of managing the registration/login/logout of a user within the application. It shows available files. It is responsible for providing torrent files corresponding to the files requested by users. It has also the task of registering users, in possession of a certain chunk/full file, in the tracker in charge of managing the corresponding torrent. If it is not present, it is responsible for determining which tracker will be in charge of handling a given torrent. *GlassFish* 5 was used as the application server.

- **Peer**: A peer obtains the list of peers in possession of at least one chunk of the file of interest through the tracker responsible for tracking this file. In particular, the peer learns the address of the tracker thanks to an interaction with the WebServer. He/She can also register within the list of users possessing at least one chunk of a file at a given tracker.

Both actions, as already mentioned, can only be performed if the peer has registered with the WebServer. The peer only needs a browser to access the platform. The actual transfer of chunks is carried out via the java client.

- **Database**: To store user credentials, information on the various files in the network and tracker information, it was decided to use the *MongoDB* document database.
- **Tracker**: Its task is to keep track of users who have at least one chuck of the file it has been entrusted with managing. Each instance has a .dets file on which to save the various chuck's owners. It is to all intents and purposes a rest server. *Here* it can be found documentation on http requests, together with examples. The server is based on *Cowboy*, http server for Erlang/OTP. The http requests have been secured by using *JWT* tokens for authentication.

Chapter 3

Platform Architecture

3.1 Website

Contains the presentation layer of the application. It contains the java server pages (jsp) and the relative servlets. The latter use the ejb-interfaces module, to perform the operations needed to perform the client request.

3.2 EJBS

There are three ejbs, two of stateless type, and one of singleton type. All ejbs implement three specific remote interfaces placed in a module called ejb-interfaces. The ejbs are the software components that, on the server side, implement the business logic of our JEE web application. They call the methods of the Dao classes in order to get access to MongoDB.

3.2.1 FilesBean and UserBean

These are the stateless ejb and are used for non-critical operations, such as create a new user, retrieve or set some user information and retrieve information of all the files contained in MongoDB. It has been chosen a stateless ejb to maximize the performance.

3.2.2 SingletonErlangBean

This is the singleton ejb and it is used for all the operations that must be thread safe such as assigning a new file to a given tracker. Singleton Session Beans are similar to Stateless Session Beans but only one instance of the Singleton Session Bean is created in the whole application and does not terminates until the application is shut down. In order to guarantee that concurrency is managed correctly the Container-managed Concurrency model has been adopted. It consists of using the lock when a thread accesses the method.
Two types of lock has been used:

1. **LockType.WRITE**: This value provides an exclusive lock to the calling client and prevents all other clients from accessing the methods marked with this annotation. The bean becomes practically a single thread. In particular it has been used for the method that is in charge of choosing the tracker that will manage the torrent for new uploaded file.

2. **LockType.READ**: This value has been used for methods such as getter ones, or method reading the database or verifying some required properties, avoiding that the bean becomes a bottleneck unnecessarily.

Chapter 4

Critical issues to be addressed

4.1 Communication problems

Communication between trackers and peers, and the web server and the trackers, is managed by means of an Erlang REST Web Service. The API that was implemented includes a GET method that can retrieve information about the peers that possess a file, given its filename, and a POST method that lets the user inform the tracker that they are sharing a file, and have it store their network parameters. Details about the API implementation can be found here.

In practice, each WebTorrent client, while downloading a file, performs periodic GET requests to the tracker that contains the file that it's downloading. This ensures that a list of peers that currently are sharing the file is always kept up to date. Furthermore, as soon as a client receives the first piece of a file, it makes a POST request to the tracker, to indicate that it is ready to start sharing the file with other peers.

Similarly, also the Web Server performs GET and POST requests towards the trackers.
Whenever a user wants to share a new file, the web server contacts the trackers with a GET request, to check whether that file is already present or not. If not, the file is added to the tracker, along with address and port of the peer sharing it. Conversely, if that file is already tracked, the peer's connection information is added to the list. In both cases, a POST request is made.

4.2 Synchronization and coordination problems

4.2.1 A peer cannot same the same chunk to more than one peer simultaneously

In order to avoid this, and any other kind of I/O-related problem, methods that need to perform read and write operations from and to a file, are explicitly defined `synchronized`. This way, we make sure that no race conditions exist during critical I/O operations.

4.2.2 The concurrent sharing of the same file by two different peers has to be handled correctly

This problem is twofold. If two users try to share the same file, and register it to the same tracker, we need to make sure that the two requests are handled sequentially, so that the entry for the specific file within the tracker is created only once, and then the second peer is added to the newly created entry. The second aspect is imposed by a design choice: each file is tracked only by a single tracker.

To ensure this property, the EJB in charge of contacting the trackers is implemented as a Singleton, i.e. only a single instance of the bean exists in the whole application at all times (this is described in more detail in section 3.2.2). This design choice enforces the atomicity of these operations.