Assignment Three

Kascade: A P2P file sharing network

Set: *19th of May 2014*
Due: *2nd of June 2014 @ 23:55 CEST*

Synopsis:
Implement a P2P file sharing client, using a centralized tracker.

Introduction

This is the third of four assignments in the *Datanet* course. The assignments are pratical in nature, and will give you you a hands-on approach to some of the technology that we all use and depend on as a part of our daily life.

In this third assignment you must implement a P2P file sharing client for the ficticious Kascade network. The protocol for Kascade is a simplified version of the popular BitTorrent protocol.

Before you start on this assignment, you should read and understand the fourth assignment as it is based on this one.

IMPORTANT: Security Note

As P2P file sharing is strongly associated with copyright infringement, the tracker used in this assignment will only allow sharing certain pre-approved files. It does not allow you to register new files for sharing, so you cannot share random files. Please use only the files listed on the tracker documentation page.

If you use your normal machine for development, please make sure that you have safeguards in your code that prevents reading or writing random files to your disk, if you choose to open your client for remote clients.

Implementation

In this assignment you will write a peer for the Kascade file sharing network. You may choose to extend your work from Assignment 2. Your peer will be interacting with other peers as well as a *tracker*. The tracker is provided for you, and you are not required to write a tracker yourself.

Data format and Protocol

The central component in the Kascade network is a .kascade file, which contains information about a single file, and the tracker that organizes it. The files are provided in *Yaml*, *bencode*, *JSON* and *XML* formats. The files contain two main

sections: file and tracker. The accepted .kascade files are found in the tracker documentation, which you can find at the link in the resources section.

The file section contains the name, kascadehash, file size, block size, file hash, and block hashes of the file. A file is split into blocks (of block-size), and each block is hashed independently with the md5 hashing algorithm. The entire file is also hashed independently of the blocks, but with the sha256 hashing algorithm. The hashes are encoded with hex-encoding (see resources) and saved in the .kascade file.

The tracker section contains a url to the tracker. Each .kascade file is identified by its sha256 hash (i.e. the sha256 hash of the original .kascade file, not the file being shared, nor the .kascade file you download).

Tracker

The tracker maintains the list of peers associated with each .kascade file, and a client can communicate with the tracker, using the HTTP protocol.

A client can participate in the network by registering with the tracker. The client sends a HTTP POST request with the following information:

- ip: The IP address the peer responds on, if omitted the current client IP will be used
- port: The port the peer is listening on
- blocks: A list of blocks the client has already, encoded in a bit-string

To register with the tracker, and obtain a list of peers, the client sends the POST message to the tracker URL found in the .kascade file. Please see the tracker documentation for more details on the request format.

The tracker responds to the request with a list of peers, which may not be all known peers. The response is a JSON array, with each element containing the information given in the PUT request, along with two extra fields: updated, and feeder. The updated field is a timestamp in in ISO 8601 format, and the feeder field is a boolean value that is set to true for peers that are known to be stable. You can use the feeder field when you develop your client, so you are certain that you get valid responses.

The bit string works by setting a bit to 1 if the peer has the block, and 0 if the peer does not have the block. The bit index corresponds to the block index. This means that for at file with 7 blocks, a peer having only block 2, 4, and 6 will construct the following bit pattern: 0010 1010. Notice that an extra bit has been added to make the length of the bit string a multiple of 8. This bit string is then encoded to hex, and results in the string `2a`.

The peer updates its progress by periodically calling the tracker with an updated PUT request, and as a part of this receives an updated list of peers, and can also track peer progress.

See the *tracker*'s homepage for concrete examples of the data formats.

Peer

You must implement the peer, using the HTTP protocol and socket libraries.

The peer must be started with a path to a local folder, and a port to listen on. It may optionally support an listening address, but should default to listening on all addresses.

The client reads all .kascade files in the local folder. For each .kascade file, the peer checks for an existing output file. If a file is found, it builds the bit string for the blocks that matches the hashes in the .kascade file. If no files are found, an empty bit string is generated. The bit string must be of length $((filesize/blocksize) + 7)/8$ bits and encoded as a hex string.

The peer communicates with the tracker as described above, and the peer sends the request as described above.

With the result from the tracker, the peer identifies other peers that have blocks that are not found locally. The blocks must either be fetched in random order, or fetched in sorted order, so the least available blocks are fetched first.

To obtain a block, the client issues a GET request to the peer ip and port obtained from the tracker. The request path is the kascadehash from the .kascade file, prefixed with a forward slash. The `Range` header is used to describe what part of the file to get. The `Range` must be a single blocksize. For the last block, the `Range` may be smaller than the block size, if the filesize is not evenly divisible with the blocksize.

After receiving a block, the peer verifies that the block hash is correct and updates the bit-string. The peer may choose to update the tracker information, or postpone the update to avoid flooding the tracker. Once all blocks are completed, the peer must verify that the file is correct, by verifying the file hash. The peer should then continue serving the file until it is shut down. The peer cannot unregister with the tracker, so other peers must be able to deal with peer errors. To avoid repeated banging on a defect peer, the implementation must perform some random selection of peers, and re-select a peer after an error. Note that some peers may send random errors, so you may want to test with only feeder peers.

If a peer recieves a GET request, it must verify that the request is valid and that it has the file being requested, as well as the block being requested. If the peer does not have the block, it must send a HTTP error message to inform the other peer. If the peer does have the block, it must serve the block, using the content type `application/octet-stream`. The peer *should* verify that the `Range` header is of correct size, and starts and ends on a block boundary. The peer *should* set the `Content-Range` header and use the status code `206` to indicate partial delivery.

Firewall and multple clients

You may not be able to open the required ports in the network filewall, or you may not want to do so. In this case, you can run a machine on the Amazon AWS network. This has the added benefit that you will not trash you local machine if you include a security hole, and it also allows you to select different geographic

regions to run your peers in. If you want to utilize this option, Amazon has granted each student a $100 gift certificate for Amazon AWS.

If you want to recieve such a gift certificate, please write an email to skovhede@nbi.ku.dk from your KU email account, or through an Absalon message, and I will grant you a gift certificate.

If you do not wish to use the gift certificate, you may use your own laptop, or use on of the shared machines that I run on AWS during the course. If you wish to do the latter, please write me at skovhede@nbi.ku.dk.

Simplifications

You may assume that no files shared are larger than `2 GiB`, i.e. that shared files fit in memory. If your machine has less than 2GiB memory, then use only the smaller files. You may also assume that blocks are no larger than `10 KiB`. You may ignore any `Connection` headers and assume `Connection: close`. You may assume that the HTTP header is always less than `4 KiB`. You do not have to download multiple blocks at a time, but you *should* be able to serve blocks while downloading.

Experiment

To evaluate the Kascade network, you must also perform some measurements and compare this to a regular HTTP request. You should conduct *at least* the following experiments:

- Compare the overall time for downloading a file via a browser versus using your client
- Measure the peak bandwidth of both downloads, and determine how soon each method reaches the peak
- Compare the CPU usage during the two downloads
- Compare the download time for the file *bbb_sunflower_1080p_60fps_normal.mp4* when downloaded with different block sizes

Resources

The tracker documentation can be found at `http://datanet2014tracker.appspot.com/documentation`.

Theory

To better understand some of the choices you have to make as a system designer, you must discuss some theoretical questions in your report. These questions are

open ended, and there is no single correct answer. The purpose is not to get the correct answer, but to make you consider alternative solutions.

The questions are:

- Is TCP the best choice for the tracker communication?
- Is TCP the best choice for the peer communication?
- Is HTTP the best choice for the tracker communication?
- Is HTTP the best choice for the peer communication?
- Can the network benefit from multicast or broadcast operations?
- How can the network utilize peers that cannot listen on a public address? (e.g. firewalled or NAT'ed peers).

Your Report

Your report *MUST* be written in ACM format. An ACM template for LaTeXand Microsoft Word is available for download via Absalon.

Your reports should contain:

- An abstract describing the contents of your report
- A description of your peer design (including libraries and frameworks used)
- A description of what experiments you have performed and their outcomes
- A description of potential bottlenecks and other issues with the network
- Answers to the theory questions

Deliverables for This Assignment

You are encouraged to work in informal groups for this assignment, for the purpose of discussing implementation details and limitations. We strongly encouraged you to come to the exercise classes, where we will use time to discuss the design, implementation etc. The implementation and report that you hand in must however be **your own individual work**.

You should submit the following items:

- A single PDF file, A4 size, no more than 3 pages, in ACM format, describing each item from report section above
- A single ZIP/tbz2 file with all code relevant to the implementation

Handing In Your Assignment

You will be handing this assignment in using Absalon. Try not to hand in your files at the very last-minute, in case the rest of the Datanet students stage a DDoS attack on Absalon at the exact moment you are trying to submit. **Do not email us your assignments**.

Assessment

The assignment will be scored on a scale of 0-10 points. There will be **no re-submission** for this assignment. In order to participate in the exam, you must obtain a total of 24 points over the four assignments.