15-440, Spring 2014
P3 Proposal
Connor Brem (cbrem), Billy Wood (wtwood)
April 15, 2014

Description

ByteTorrent will be a BitTorrent-esque file-sharing protocol. Our service will contain two node-types: Clients and Trackers. Each file will be broken into chunks, and the Tracker will maintain a list of which clients have which chunks.

Clients will connect to each other, and will attempt to trade chunks that they have for chunks that they want. As with BitTorrent, Clients will not connect to all other Clients. Rather, each Client will connect to a small number of neighbors, suggested by the Tracker. If a Client is unable to get a chunk from its neighbors, it will ask the Tracker for another Client that it can connect to.

Structure

Client

The Client nodes will be implemented in Python. They will have the following abilities:

- Create new .torrent files and send them to a Tracker
- Given a .torrent file, connect to a Tracker identified in the .torrent file and receive instructions.
- Serve parts of a file to other Clients on request
- Inform Tracker when it has a piece
- Report themselves when they do NOT have a piece that other clients think they do
- Contact Tracker when none of their neighbors have a desired piec.
- Connect to a new Tracker when the current connection dies.

Tracker Server

The Tracker nodes will need to be highly distributed. As such, we've decided to implement Trackers in Go. The Trackers will be able to:

- Receive new .torrent files from Clients
- Serve .torrent files to frontend
- Keep .torrent files updated
  - Add/remove Clients from list of Clients with torrent pieces
  - Get new .torrent files from frontend

Tracker Frontend

We ultimately aim to add a Frontend for the Trackers. This will be done in Javascript and HTML. Its purpose will be to:

- Search for torrents by name
- Fetch .torrent file from backend for user to download
- Send new .torrent file to backend from user

Distributed Systems Concepts

Clients

The Client nodes will be in constant contact with each other. They will need to be robust to neighbors dying, and Trackers dying. We plan to use exponential back-off to disengage from unresponsive connections.

Trackers

All Tracker servers will maintain the same data. We will implement this with Paxos.

Test Plan

Tracker Tests

- Kill a Tracker in the quorum, and make sure that things can still get done.
- Kill a Client, forcing the Trackers to update the database.

Client Tests

- Disengage from an unresponsive neighbor in the Client web.
- Connect to a new Tracker if current one becomes unresponsive.
- Ask a Client for a chunk that it does not have.

Development Tiers

Minimum Testable Product

- Client:
  - Create .torrent files and send them directly to one tracker
  - Get pieces from other clients, given a .torrent
  - Serve pieces to other clients
  - Can fail without disrupting things
- Tracker Server:
  - Single tracker (no Paxos)
  - Record when there are new files
  - Record when new clients get pieces of files

Target Deliverable

- Client:
  - Self-report to tracker when client does not have a piece that other clients expect it to
  - Compute/create checksums on torrents to ensure security
- Tracker Server:
  - Replicated via Paxos

Stretch Goal

- Client:
  - UI: Show currently downloading/seeding files
- Tracker Server:
  - Maintain users/manage logins
- Tracker Frontend

Development Schedule

Week 1: April 14-20

For Week 1, our goal will be to have the Minimum Testable Product prepared.

- Client:
- Tracker:

Week 2: April 21-28

For Week 2, we will focus on getting our project to the Target Deliverable state.

Week 3: April 29-May 1

In Week 3, we will work on the Stretch Goals. In particular, this means working on the GUI.