A Distributed, Peer-to-Peer File Backup Service

https://github.com/DistBackup/dbscore
Benjamin (Benji) Holen (benjamin.holen).edu
Brinley Macnamara (brinley.macnamara@tufts.edu)
Xuanrui (Ray) Qi (xuanrui.qi@tufts.edu)

Outcome Analysis

We successfully achieved our minimum deliverable, which is a fully functioning peer-to-peer file backup service which allows uploads and downloads of files.

However, we were not able to achieve our maximum deliverable, from which we were mainly missing encryption. The main reason for this omission is that encryption makes debugging extremely difficult.

Design Reflection

Reflecting on our design, our design was mainly on the right track, and also allowed us to maximize our utilization of Erlang/OTP's built-in support for distributed computing architectures. However, the TCP protocol was significantly more complex than Erlang/OTP's interprocess communication protocol, and a higher-level abstraction over TCP would ease the work.

Division of Labor

Our team members were able to work independently of each other, but we still needed to meet often and integrate our implementations and modify them if required. Each member was responsible for implementing different elements of our program:

- Brinley: gen_tcp servers, monitor, and client interface
- Ray: file-handling (i.e. reading, hashing, and writing files)
- Benji: databases

Bug Report

There are no known bugs in this version. Since there was no major architectural flaws from the beginning, most difficult bugs proved to be misuse of variables (which could be difficult to spot, and sometimes did not cause the program to fail) and networking issues. The omnipresence of firewalls and large number of used TCP ports can fail networking easily, but is very hard to debug.

Overview of Code

Our code is written in pure Erlang. It can be divided into four sections: client, server, monitor and utility. Client and server both run on each peer in the cluster, and monitor runs only on the monitor of that cluster:

Client:

- start_client.erl: starts the client
- client.erl: main functions for the client

Server:

- tcp_server.erl: the TCP server
- tcp_sup.erl: the TCP supervisor for client's server

Monitor:

- start_monior: starts the monitor
- monitor.erl: main functions for the monitor
- monitor_tcp_server.erl: the TCP server for monitor
- monitor_tcp_sup.erl: the TCP supervisor for monitor's server

Utility:

- file_proc.erl: utility for file and file packets
- database.erl: high level interface for the dets database

How to run our program:

1. ssh into 3 lab machines (at minimum, our app needs one monitor and two clients (peers))
2. cd into our project directory
3. Choose one machine to be the monitor, and find the public ip address of this machine with:

   ```
   dig +short myip.opendns.com @resolver1.opendns.com
   ```

   E.g.

   ```
   resnet143-22:~ brinley$ dig +short myip.opendns.com @resolver1.opendns.com
   130.64.143.22
   ```

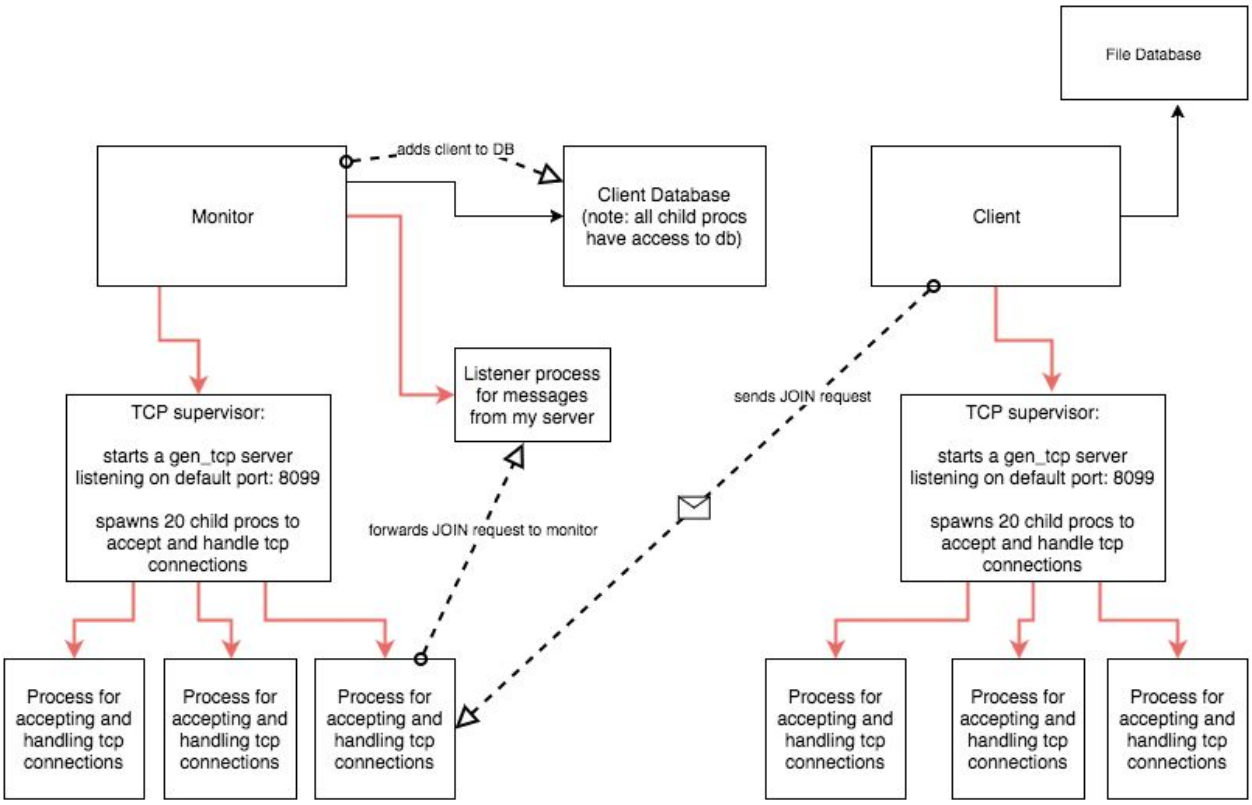
4. Open ip.conf and replace the existing IP address with your monitor's IP address. Save and quit.
5. Start an erlang shell on each lab machine with:

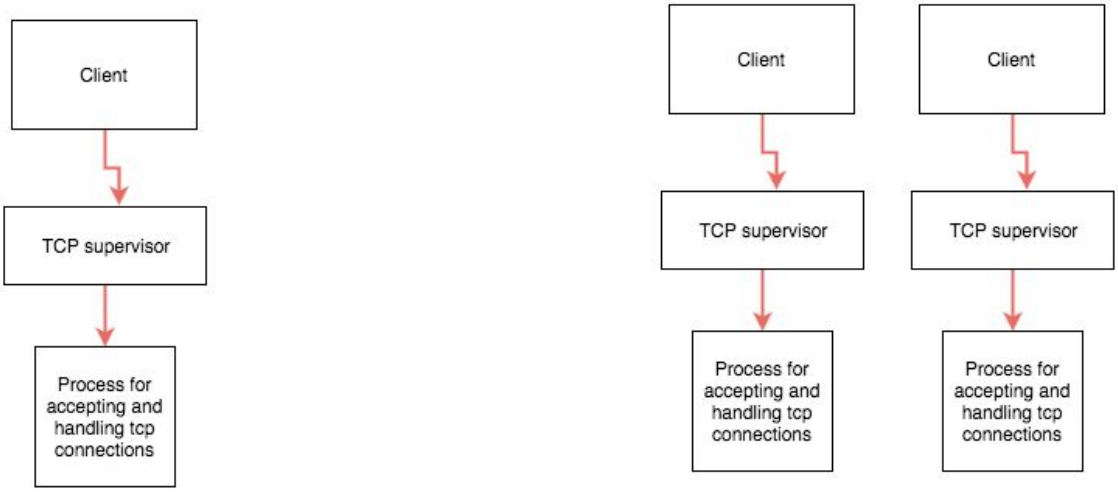
   ```
   erl -name your_node_name -setcookie your_cookie
   ```

6. In one erlang shell, compile our code with: `make:all().`
7. Start a monitor on the machine you've designated as the monitor with: `start_monitor:start().`
8. Start clients on the other machines with: `start_client:start().`
9. Press enter on the client machines for a list of public functions

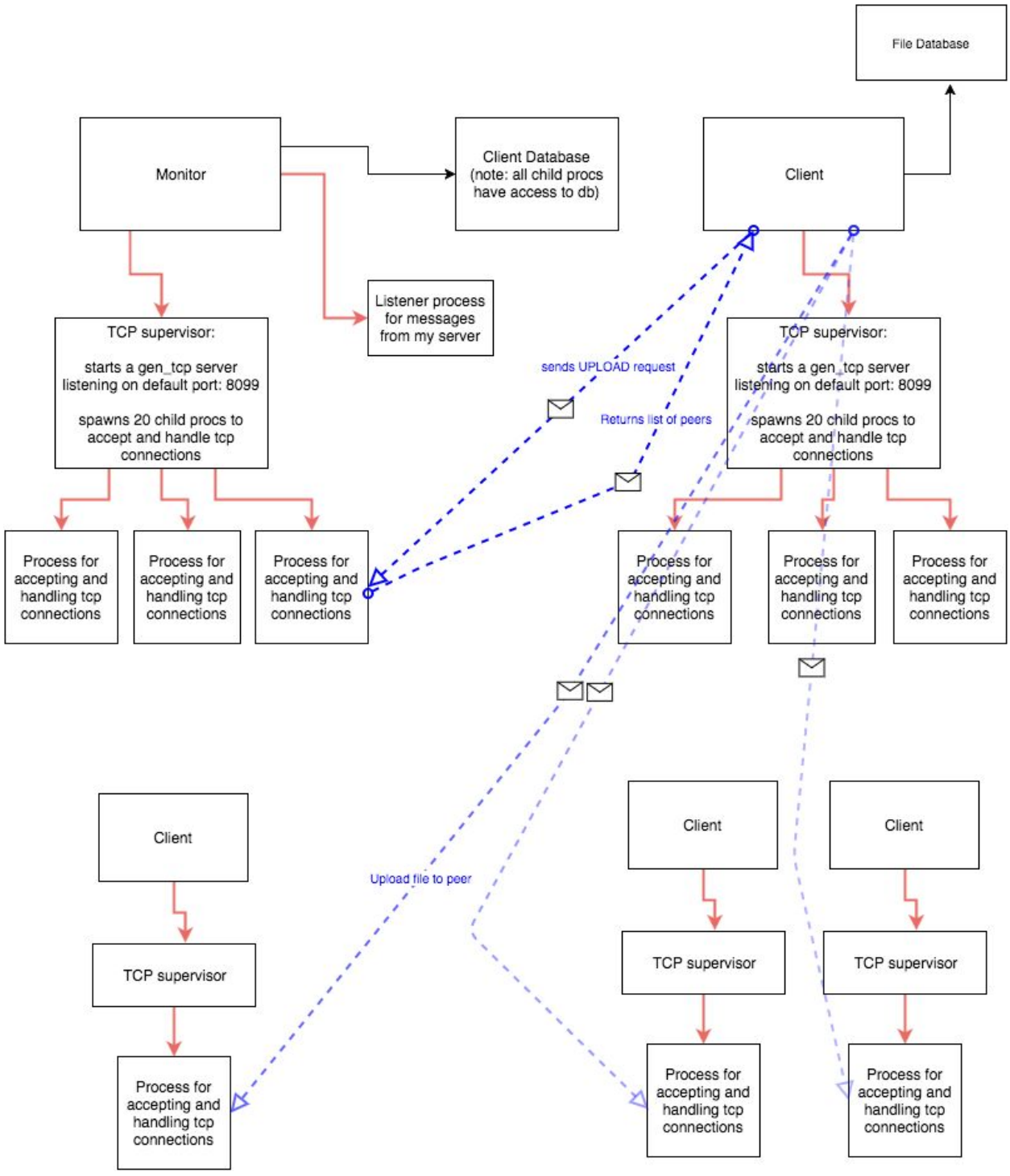
Final Refinement of Design

Join:

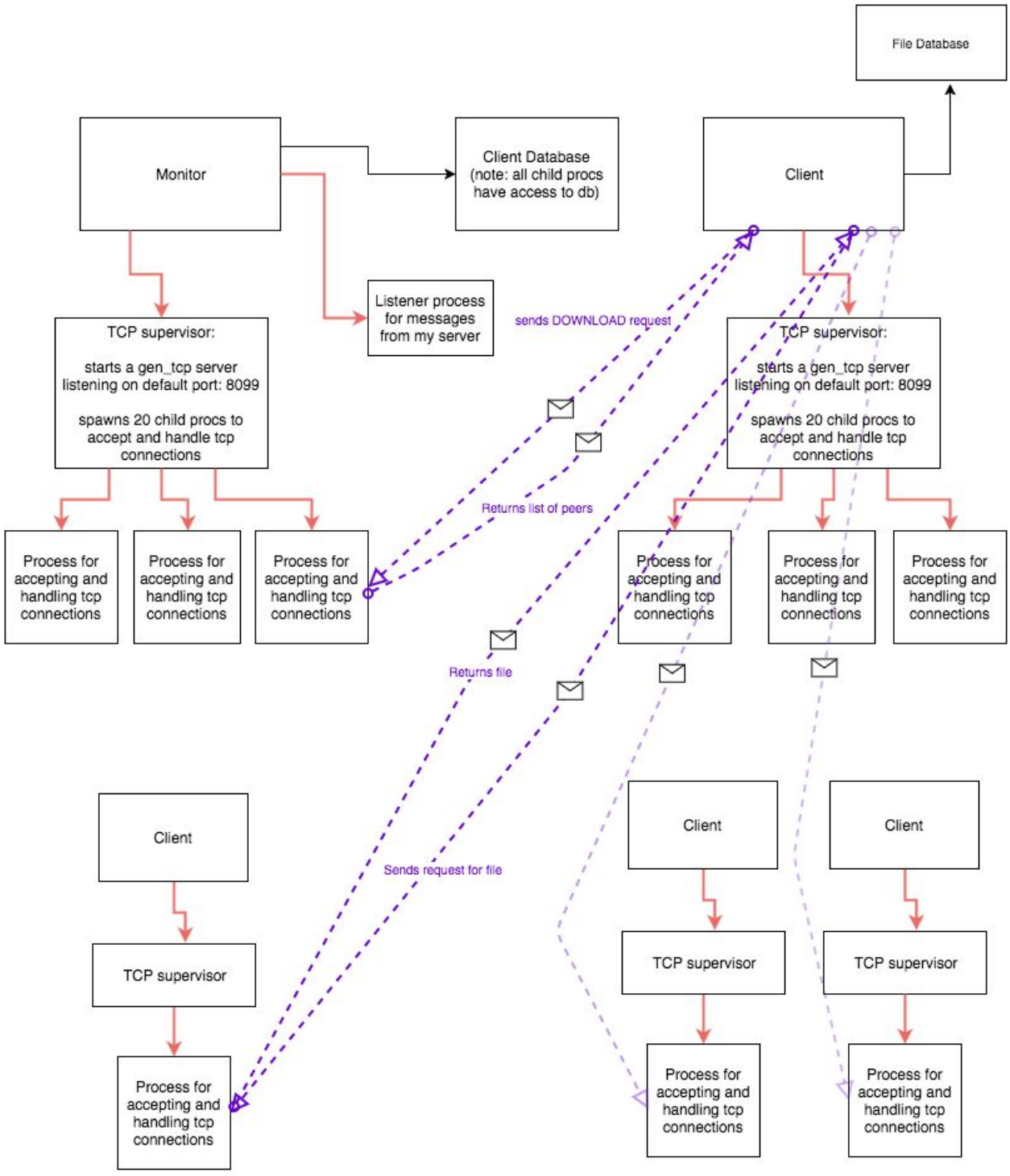




Upload:


File Database
Monitor
Client Database
(note: all child procs
have access to db)
Client
Listener process
for messages
from my server
TCP supervisor:
starts a gen_tcp server
listening on default port: 8099
spawns 20 child procs to
accept and handle tcp
connections
sends UPLOAD request
Returns list of peers
TCP supervisor:
starts a gen_tcp server
listening on default port: 8099
spawns 20 child procs to
accept and handle tcp
connections
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Client
Client
Client
Upload file to peer
TCP supervisor
TCP supervisor
TCP supervisor
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections

Download:


File Database
Monitor
Client Database
(note: all child procs
have access to db)
Client
Listener process
for messages
from my server
sends DOWNLOAD request
TCP supervisor:
starts a gen_tcp server
listening on default port: 8099
spawns 20 child procs to
accept and handle tcp
connections
TCP supervisor:
starts a gen_tcp server
listening on default port: 8099
spawns 20 child procs to
accept and handle tcp
connections
Returns list of peers
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Returns file
Client
Sends request for file
Client
Client
TCP supervisor
TCP supervisor
TCP supervisor
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections
Process for
accepting and
handling tcp
connections