Distributed systems, University of Helsinki, Fall 2022

Design plan

Distributed Systeam – Team members

Sami Toiskallio
Hiski Ruuska
Nicholas Sakyi
Joonas Jakobson

Selected topic

The goal of the project is to implement a peer-to-peer decentralized chat application. The main focus is the networking side of things, while the UX of the chat application is secondary. The design should minimize single points of failure. The initial bootstrapping of new nodes joining the network should not be handled by a single node. No logs are collected.

Description

The nodes in the system form an overlay network (DHT – Distributed Hash Table), where a single node knows the IP address of only *log(n)* other nodes, not all. We have chosen to use the Chord implementation of DHT.

The DHT is used to "store" the details of known users. Each time a node joins the network, the user specifies a username (username per session). An id is generated for the node automatically without interaction from the user. In the hash table the key is of form **hash(username#id)** and the value is the corresponding **IP address**. The combination of username and an id makes it highly probable that all users can be identified uniquely without communication (checks) between bootstrap nodes. The user details should be "deleted" from the DHT when the messaging session(s) of the user end.

The connection between nodes will be handled by the python-p2p-network**[1]** python library. The library uses the TCP/IP stack to hold the connection between nodes.

The security of the communication is out-of-scope for this project. In the future the communication could be secured by public key cryptography.

Node roles and functionalities

There are two types of nodes; bootstrap nodes and messaging nodes. The bootstrap nodes handle the new nodes joining the networks for the first time. The messaging nodes who have joined to the network can form direct P2P chat connections with each other. All nodes participate in the DHT.

The IP addresses of the bootstrap nodes are hard coded to the application source code so that new nodes can join the network without knowing any peers initially. The bootstrap nodes are hosted by the developers (us).

Messages

- **Join message sent to the bootstrap node**
    - Includes the details of the joining messaging node
- **Messages related to the functionality of the DHT**
    - **ping_predecessor** (is it still alive? If not alive, mark it “null”)
    - **notify** (if the predecessor of some other node is null, notify the node that I might be predecessor)
    - **stabilize** (ask the successor node if “I am” its predecessor)
    - **find_successor** (ask the successor node of a key value [node])
- **Text messages between messaging nodes**
    - python dict {“message”: “Hi there!”}

The *ping_predecessor*, *notify*, and *stabilize* messages “repair” the DHT in the events of nodes crashing, leaving or joining the network.

Features

- A user can select a username for each session
- A user can view their id
- A user can find other users by their username#id
- A user can send a message to another user (establish a chat session)
- A user can end the chat session with another user

The application is scalable. The upper limit of the users is defined by the hashing function in the DHT. For example, if SHA-1 was used, then the upper limit of users is 2^{160} (SHA-1 generates a 160 bit digest).

Fault tolerance is achieved by the structure of the DHT. A node can crash and the system will recover. The crashing node is marked as “unknown” by its successor and it will be replaced by another node after a while. The messages related to the functionality of the DHT are sent periodically.

References

[1] python-p2p-network, @macsnoeren
https://github.com/macsnoeren/python-p2p-network