Project report: Stage 1
COMP3100 Distributed Systems, S1, 2023
HMClient - A Client-side Scheduler
SID: 47083573, Name: Harris Coughlan McRae

1 Introduction

The aim of this project — HMClient — is to create a vanilla client-side simulator of a distributed system setup. By connecting to a pre-made server simulator[1], an analogue of a distributed system can be emulated by receiving "jobs" from the servers, and in response, scheduling these "jobs" to a number of connected "servers".

This project in particular will be running a basic scheduling algorithm — Largest Round Robin.

The project can be found at the following link)[2].

2 System Overview

The system consists of two components:

- a server simulator[1] that will produce a list of servers and jobs, and allow a client to connect to the server, and;
- a client simulator that will connect to a server, receive jobs, and allocate them to servers as per the used algorithm.

2.1 Server Simulator (DS-Server)

The ds-sim server[1] is a program emulating the scheduler of a distributed system. The server will open a network for a client to connect to, and generate a number of jobs and servers.

There are multiple types of server, with each type of server having a number of cores, an amount of RAM, and an amount of disk space.

Jobs are generated with the 'minimum specs' needed to run the job - a number of cores, an amount of RAM and an amount of disk space necessary to complete the job.

The server, after connecting to the client simulator and authenticating, will provide a job listing to the client, and wait for the client to respond & schedule the job to one of the servers.

2.2 Client Simulator (HMClient)

The client simulator[2] is modeled after the client component of the aforementioned ds-sim program(s). It connects to the open server mentioned earlier.

Once connected, the client will perform a handshake with the server & authenticate itself. It then anticipates receiving a job to schedule.

When a job is received, the client will check the work servers accessible to the server, and determine the largest server type (based on the number of cores). Jobs will be scheduled only to the largest server type in a pattern referred to as 'Largest Round Robin' (LRR) — each job is scheduled to the next server in the largest server type group, wrapping around in a circle until completion.

The client is responsible for doing all the processing and responding to the server, with the server acting as a messenger for each job & server — the client will have to respond appropriately to prompts from the server including:

- jobs to be scheduled
- completion of scheduled jobs
- end of job backlog
- edge case messages & authentication

Additionally, the client simulator will have limited customizability, in the name of future-proofing. This includes the ability to specify IP addresses to connect to, port numbers to utilize, and whether or not it will log all messages it receives. In future renditions this will be expanded to allow for changing of algorithms and other features.

3 Design

HMClient is intended for forwards-compatability — the ability to continually upgrade and modify the program as requirements change. This open-endedness in intention requires similar design choices to be made.

Originally, HMClient operated on an ambiguous state-machine architecture — the program flows through a number of states, with state changes triggered based on the message received and the current state. However, this had been removed for clarity, though without sacrificing the ability to expand upon the current design later.

The current design emphasizes clarity. The program executes in a small number of discrete steps, with each one clearly documented.

The choice to minimize unnecessary steps in execution was also made early on, similarly to make the program easily understandable. Since only one algorithm is necessary & accessible at this point in time, it can take place of the main program, instead of using a more modular state machine design which adds unnecessary complexity. Some small complexity was added in the form of command-line arguments used to slightly customize the program, but not to a great extent.

4 Implementation

HMClient was programmed in Java, utilizing native Java libraries for standard IO and networking, namely socketing.

HMClient relies on a buffered reading of messages in more plaintext format, using linebreak characters instead of null characters to terminate a string. In the future this may be customizable. Each instance of HMClient (a class in the program) has its own unique socket, reader and writer, so that theoretically multiple schedulers may be up at one time.

There are no notable data structures in the program bar the HMClient instance itself. The process of determining the largest server for use in the LRR algorithm is performed as the messages are received from the server itself. The process is as follows:

```
let LargestServer = undefined
let CoreCount = -1
let ServerCount = -1

for each message in buffer

    split message into array words

    if words[4] as int is greater than CoreCount, do:
        set CoreCount to words[4] as int
        set ServerCount to 0
        set LargestServer to words[0]

    if words[1] is equal to LargestServer, do:
        increment ServerCount by 1
```

This process negates the need to store the total list of servers, and instead will end up storing the largest server type, the number of servers of that type, and the number of cores it has. It also ensures that only the first server type with the largest number of cores is utilized.

The program has a few wrapper functions to greatly ease the complexity of the main code; namely, a few functions that obscure the process of reading messages from the server or writing messages to the server into one single statement. This allows for much easier modification and reading.

References

[1] Y. C. Lee. "ds-sim". URL: `https://github.com/distsys-MQ/ds-sim`.

[2] H. C. McRae. "COMP3100 (HMClient)". URL: `https://github.com/hmc-rae/COMP3100`.