Project report: Stage 1

COMP3100 Distributed Systems, S1, 2022

SID: 46564616, Name: Weibin Chen

[1]

1 Introduction

A Distribute system consists of various sizes of autonomous computing nodes. Each node collaborates with the other but appears as a single coherent system to end-users. The simulation is based on the client-server architecture. A client program can understand and obtain information from servers and then distribute their work depending on demand and available resources, making efficient use of computing resources to perform as efficiently as possible.[2]
The first phase of this project focuses on the client-side when a plain version of the client simulator (called ds-client). It is designed and implemented, based on the given job dispatching and job scheduling resources, developed using Java, Ubuntu OS, Wireshark, GitHub completes the collaboration. The client connects to the server-side emulator, receives jobs, and schedules them according to the server type with the most resources available.

2 System overview

The system consists of two parts, the server-side and the client-side. It is based on the ds-sim distributed system simulator. In the current design, both the client and the server are located in the same host, which can easily set up file exchange without involving complex file exchange protocols but does not affect the simulation effect and experimental purposes.
The server listens for potential connections on the specified TCP port (TCP port 50000). When the client initiates a link to the server, a connection between the client socket and the server socket establishes message communication.
Both sides go through a handshake, initialize communication and implement a loop to do the work.
Clients must follow the correct canonical message form described in the ds-sim user guide. Therefore the client must follow the handshake protocol before the client can receive jobs to be scheduled. The server creates a task list and transmits it to the client. When the client gets a job request, the client can send a GETS message and parse the data from the system information XML files. According to the current scheduling policy, the priority of the task, then the client determines which server is suitable for the current job and submits that job to the appropriate server. The client will then enter a loop that will keep listening for jobs that need to be scheduled or any other messages the client will process. It will happen until the client indicates a NONE message is received that there are no more jobs to schedule. Then both the client and the server can exist from the simulation.

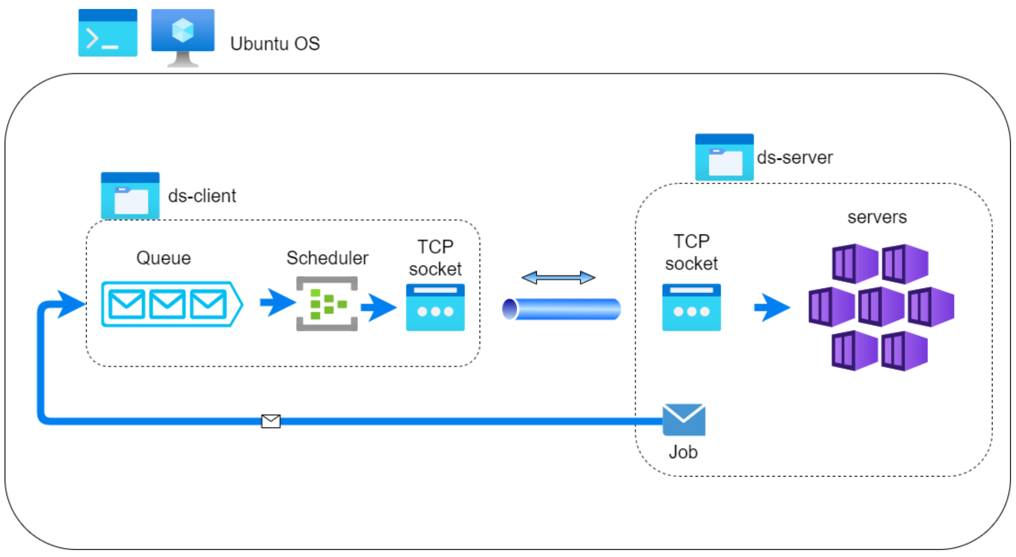


Figure 1: ds-sim structure

3 Design

Regarding the project's design concept, we have instilled the principles of modularity and readability to make the client simulator a modular system. It process communication data and perform algorithm analysis.
We break down the project step by step, into each simple-to-understand, easily implementable step. Make every new generic function can be created and easily plugged into the client program for future rapid development, enabling the design of various components and understanding. We will ensure that modularity is allowed to continue as the solution is built as the system evolves.

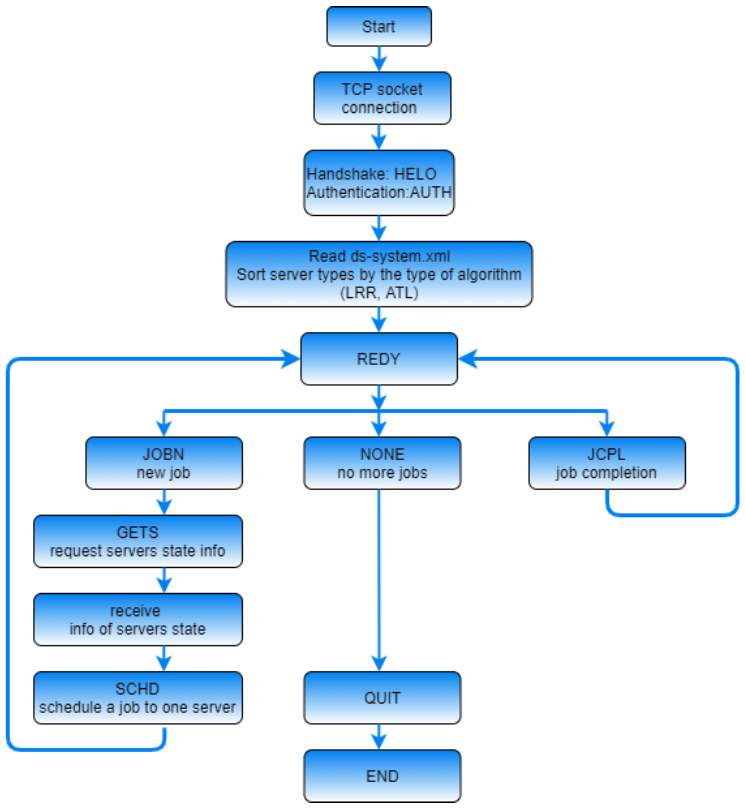


Figure 2: ds-client flow chart

3.1 State machine

The scheduler uses a switch-based control scheme. The use of state loops allows code reuse and scalability in various situations. The switch-based design works well with the help of while loops because it provides specific code branches to be executed in each loop. Using a case-break branch also allows the code to be more readable, as indentation and brevity are better than if-else designs.
The client program will track multiple similar different states receive various commands from the server. The scheduler uses a switch-based control scheme that uses state to determine position in the communication protocol. For the same command, it will respond differently according to different server states and algorithm requirements.
We used the switch statement and the Enum keyword to deal with multiple cases of different states. Some more complex cases, such as receiving information about the available statement of the servers, or scheduling a job, require additional functionality to be done.
The handshake protocol is one of the most basic functions in a client-server architecture, and it ensures that the client and server establish a correct data connection with each other. The handshake method starts with a HELO message sent from the client to the server, and then the client must authenticate itself with the server. Once authenticated, the server creates a ds-system.xml file that lets the client know about the server information and is ready to start scheduling the job. The server will receive a REDY message, the client tells the server that it is prepared to schedule the jobs, and the server will send its first job. All the above processes are programmed based on the state machine.

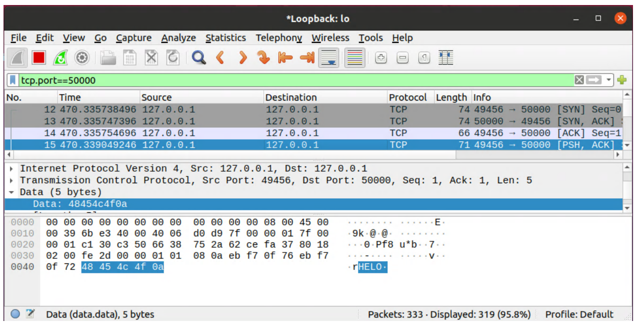
3.2 Algorithms

In the job scheduling system, the essential functions of the emulator are mainly composed of three parts: job, server, and client.
The Client uses a Largest-Round-Robin algorithm (LRR) to schedule jobs. It queries the ds-server XML file for the largest server type. After receiving a job in this system, the client identifies the number of largest server types with GETS command. Then it sends the job to the currently largest active servers in a round-robin fashion.

4 Implementation

In the general implementation of the project, the following essential software is used to build the simulation environment and carry out software design.

- Ubuntu-20.04.4-desktop: Ubuntu is a Linux-based operating system, the operating system required by ds-sim.
- VirtualBox-6.1.32: Develop a separate Linux virtual machine based on the Windows computer environment without affecting the original PC environment.
- OpenJDK-17-JDK: is the java compiler that allows the use and creation of Java-based code files.
- MobaXterm V21.2 is a remote network tool that provides a tabbed terminal with SSH in windows.
- WinSCP V5.13: its primary function is file transfer between windows and ubuntu.
- Eclipse IDE-20210612
- Wireshark: It can capture the real TCP traffic between the server and the client when debugging code.


*Loopback: lo
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.port==50000
```

```
No.  Time          Source     Destination  Protocol Length Info
  12 470.335738496 127.0.0.1  127.0.0.1    TCP        74 49456 → 50000 [SYN] Seq=0
  13 470.335747396 127.0.0.1  127.0.0.1    TCP        74 50000 → 49456 [SYN, ACK]
  14 470.335754696 127.0.0.1  127.0.0.1    TCP        66 49456 → 50000 [ACK] Seq=1
  15 470.339049246 127.0.0.1  127.0.0.1    TCP        71 49456 → 50000 [PSH, ACK]

› Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
› Transmission Control Protocol, Src Port: 49456, Dst Port: 50000, Seq: 1, Ack: 1, Len: 5
⌄ Data (5 bytes)
    Data: 48454c4f0a

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 39 6b e3 40 00 40 06  d0 d9 7f 00 00 01 7f 00   ·9k·@·@· ········
0020  00 01 c1 30 c3 50 66 38  75 2a 62 ce fa 37 80 18   ···0·Pf8 u*b··7··
0030  02 00 fe 2d 00 00 01 01  08 0a eb f7 0f 76 eb f7   ···-···· ·····v··
0040  0f 72 48 45 4c 4f 0a                               ·rHELO·

Data (data.data), 5 bytes        Packets: 333 · Displayed: 319 (95.8%)    Profile: Default
```

Figure 3:

- Git-2.34
- GitHub
- TortoiseGit-2.12: it is a powerful windows shell of GIT. It collaborates with git and GitHub to handle version control functions.

The realization of the virtual job scheduling system is carried out through multiple java files, each of which undertakes the specified tasks so that the system can work as designed. A total of six java files are created, of which the file DsClient.java performs all job scheduling tasks, and the rest of the files are data structures and public variables. The six files created are as follows:

- DsClient.java
- Commands.java
- JobSubmission.java
- ServerType.java
- ServerStatus.java
- Statement.java

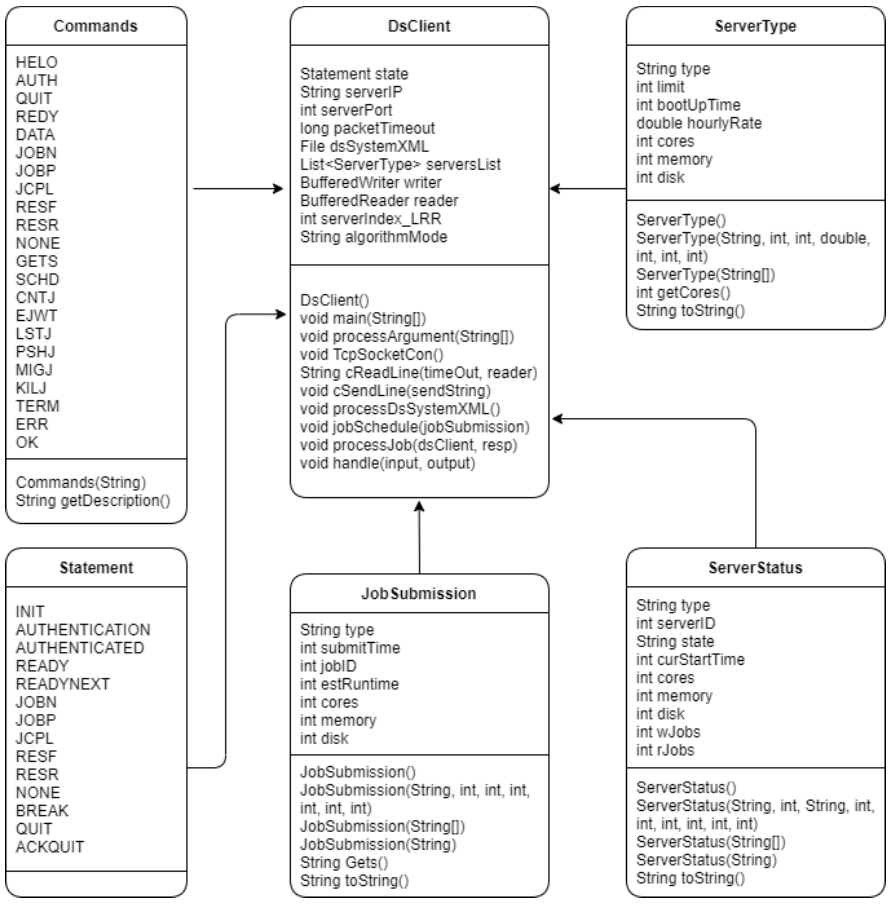


Figure 4: ds-client java UML.

4.1 Data structure

There are a lot of information and commands. They are sent in a String stream with simple keywords and numbers. It isn't easy to understand and do the data process in the further steps. We can format the data stream with class structure and store them in an arraylist.

- Read stream for xml file and sort them by CPU cores

  Format ds-system.xml with ServerType class:

  type:t2.aws,limit:1,bootUpTime:60,hourlyRate:0.8,cores:16,memory:64000,disk:512000

  type:t1.medium,limit:2,bootUpTime:40,hourlyRate:0.4,cores:4,memory:16000,disk:64000

  type:t1.micro,limit:2,bootUpTime:30,hourlyRate:0.2,cores:2,memory:4000,disk:16000

  type:t1.small,limit:2,bootUpTime:40,hourlyRate:0.4,cores:2,memory:8000,disk:32000

- Receive a job submitted by the ds-server

  JOBN 32 0 47066 1 700 600

  After format with JobSubmission class.

  type:JOBN,submitTime:32,jobID:0,estRuntime:47066,cores:1,memory:700,disk:600

- Receive the entire server state information from ds-server

  t1.micro 1 inactive -1 2 4000 16000 0 0

  t1.small 0 inactive -1 2 8000 32000 0 0

  t1.small 1 inactive -1 2 8000 32000 0 0

  t1.medium 0 inactive -1 4 16000 64000 0 0

  t1.medium 1 inactive -1 4 16000 64000 0 0

  t2.aws 0 inactive -1 16 64000 512000 0 0

  After format with ServerStatus class:

```
type:t1.micro,serverID:0,state:inactive,curStartTime:-1,cores:2,memory:4000,disk:16000,wJobs:0,rJobs:0
type:t1.micro,serverID:1,state:inactive,curStartTime:-1,cores:2,memory:4000,disk:16000,wJobs:0,rJobs:0
type:t1.small,serverID:0,state:inactive,curStartTime:-1,cores:2,memory:8000,disk:32000,wJobs:0,rJobs:0
type:t1.small,serverID:1,state:inactive,curStartTime:-1,cores:2,memory:8000,disk:32000,wJobs:0,rJobs:0
type:t1.medium,serverID:0,state:inactive,curStartTime:-1,cores:4,memory:16000,disk:64000,wJobs:0,rJobs:0
type:t1.medium,serverID:1,state:inactive,curStartTime:-1,cores:4,memory:16000,disk:64000,wJobs:0,rJobs:0
type:t2.aws,serverID:0,state:inactive,curStartTime:-1,cores:16,memory:64000,disk:512000,wobs:0,rJobs:0
```

4.2 Functions

4.2.1 cReadLine()

TCP client readLine() blocking function A complete service may be split into multiple packets by TCP for transmission, or numerous small packages may be encapsulated into one large data packet. It is a problem of TCP unpacking and packetization.
The receiver needs to follow the same convention as the receiver to determine the start and end of the message. In this experiment, we agreed to add a line feed character as the message separator at the end of each message. On the server-side, we used the -n parameter. On the client-side we used the readLine() function.
But because the readLine() function is a blocking function. If there is no further reply due to some program code debugging errors, such as the server cannot understand the command we sent. Because no valid reply data is received, the readLine() function will block the entire main program, causing the program to fail as a ”Crashing phenomenon.” We can't perform multi-thread and multi-thread data exchange, so add a timeout function before the readLine() function is executed and uses the cReadLine() function to wrap the readLine() function, making it a nominally non-blocking function. The state mechanism can continue to operate.

4.2.2 processDsSystemXML()

Read XML file
XML is a tree-structured extensible markup language, starting at the root and extending to the leaves. Steps to parse XML using DOM:

1) Create an object of DocumentBuilderFactory.
2) Create a DocumentBuilder object.
3) Obtain the Document object through the DocumentBuilder.parse() method.
4) Get the list of nodes through the getElementsByTagName() method.
5) Iterate over each node through for a loop.
6) Get the attributes and attribute values of each node.

Java 8 introduced enhancements to the Comparator interface. Comparators can be passed to a sort method (such as Collections.sort or Arrays.sort) to allow precise control over the sort order.[3]

```
Collections.sort(serversList, Comparator.comparing(ServerType::getCores).reversed());
```

All we can do is sort the server types by CPU cores. The largest type of server with the most CPU cores will move to the ArrayList's top.

References

[1] C. Weibin, "Ds-sim-client." [Online]. Available: https://github.com/benbandbetter/DS-SIM-CLIENT

[2] M. Steen and A.S.Tanenbaum, *Distributed Systems*, 3rd ed. distributed-systems.net, 2017, ch. 1-2, iSBN: 978-90-815406-2-9.

[3] Oracle, "Java™ platform standard ed. 8," 2022, accessed Mar, 2022. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html