Efficient server use through distributed computing based on job scheduling

Seonghun Park (44401213)
Louise Easterbrook (45236275)
Randle Valerio (45952507)

Introduction:

The increasing use of computers across the population has increased the computational problems' size that the system has to deal with. One of the methods that came out to solve this problem is a distributed system. Andrew S. Tanenbaum's book Distributed System Concepts, a textbook on distributed systems, defines a distributed system as a set of independent computers that look like a single system. In other words, each computer that exists as a separate entity is interlocked on one system and merged into a vast independent system. The above definition can be viewed from two main perspectives. One is a hardware machine that runs independently and autonomously, and the other is software, a system that is considered as a system from a user's point of view. These two points are the core elements of a distributed system and are a principle that must not be forgotten when designing a system. The most important thing when designing a distributed system based on the two major principles above is the process of effectively distributing huge computational problems by software that combines various hardware machines (servers) into one. Through the process of calculating the size of resources required for each calculation task and assigning a new job based on the current server load and future schedule, you can maximize the calculation speed or increase server stability by using all servers to the maximum. We will develop a server and a user client through this project that can meet various scheduling algorithms, assign jobs to distributed servers, and plan schedules. First, we plan to develop a client site simulator with basic scheduling and job distribution capabilities in the initial vanilla version.

System overview:

Currently, the system is largely composed of three parts: Handshake Process, Receiving Loop State, and Scheduling State.

Handshake Process:

The handshake Process establishes the connection and basic communication between the server and the client. First, we declare the DataOutputStream and BufferedReader over the socket. Then the client sends a HELO message from the server, and the server sends OK. After that, the client's system recognizes the user name through (System.getproperty("user.name")) and sends it to the server. After the server sends OK again, the ds-system.xml file containing the server information is sent to the client. The client uses the information from this server to find the largest server.

Receiving Loop Process:

In this loop step, the server that has finished the handshake waits for input such as a job or command from the client. This loop has 'while(true)' as a condition, so the loop repeats until it encounters the break statement. At this stage, the server breaks the loop through the NONE command or starts scheduling the job given through the JOBN command.

Scheduling State:

First, in the scheduling state, the process of finding the largest server through XML Parsing proceeds. Client finds the server with the highest core count in the ds-system.xml file provided through XML parsing. After that, the server checks how many lines of code will be given to the server through the DATA[Integer] command. The client then checks through a for loop to see if the server is reading the correct amount of rows. When this is confirmed, the client checks the server type and ID and sends the Job ID through the SCHD command. When all of this is done, we reach the end of the loop and go back to the first statement of the job dispatcher and repeat that process. If the iteration process completes the task, the server sends a JCPL command to the client to inform the client that the task is complete. In the process, the client also gets out of the loop and sends a QUIT to the server. Then, when the client receives a message from the server, it closes the socket opened through handshake and the DataOutputStream and BufferedReader.

Design:

Distributed System is largely divided into a scheduler that determines the work to be allocated by utilizing the resources of the server, and a dispatcher that actually allocates work to the server. When the Dispatcher allocates jobs to servers without any special notice, the efficiency of the system is determined by how efficiently the scheduler distributes jobs to various servers. Currently, this design contains only simple functions for connection

between server and client via handshake and basic job allocation. All jobs on the client are allocating all jobs to only one server with the largest core count among many servers. This is a very inefficient algorithm. As all jobs are assigned to one server with the largest quota, a large burden is placed on only one server, and servers with other free quotas will not be granted jobs while that server is finished with the job. To solve this problem, we need to create a scheduler based on a higher level algorithm. Scheduling algorithms have different techniques, but different types of systems are preferred. In the case of FIFO (First In First Out) scheduling, which is the simplest scheduling technique, it is a non-preemptive scheduling method in which a CPU is first allocated to a task that has entered the standby queue first. Non-critical tasks can keep you waiting for important tasks, which is also called First Come First Served scheduling. Priority scheduling is a method in which a priority is given to each task, and the CPU is allocated to the task with the highest priority first. Tasks with low priority may fall into Indefinite Blocking or Starvation, and as a solution to this, an aging technique that increases the priority according to the residence time can be used. First, we need to determine the scheduling method to be introduced into the system based on the status and type of the job currently allocated to us, and the quota of the server. The introduction of a new scheduler will solve the inefficiencies of the current system and the problem of overloading the quota per serving.

Implementation:

The Project's Implementation used a variety of resources to meet the requirements and in order to achieve the goal of scheduling all the jobs to run at the largest server.

There were a variety of technologies that were needed for the purpose of completing the project. Firstly, the Virtual Machine Virtual Box is a software that emulates or virtualizes a different operating system. The Virtual Machine Virtual Box was used to run a Linux operating system called Ubuntu. Ubuntu is the main operating system that was operated in order to implement and test the source code of the project. GitHub is a hosting platform for a collection of different kinds of source codes. It allowed our group members to work together on the project in any place we may be. First, we created a repository on GitHub to store all the files for our project, then each of the group members cloned a local repository on each of our devices. In the terminal, Nano is our primary text-editor where we physically wrote the source code for the client java file for the project.

In terms of techniques that were used, variables and loops were the main techniques that were often used within the implementation of the source code. Multiple string variables were created in order to store the messages that were being sent by the server. The first loop was created in order to schedule each job that the server wanted to be scheduled, the loop only ends when there are no jobs that need to be scheduled. The nested loop was created in order to find the largest server in the given xml file.

In terms of data structures, arrays were used in order to store the messages of the jobs, servers, etc., sent by the server.

There were a variety of libraries that were needed for the client source code to complete its tasks of scheduling jobs such as the java.io, java.net, javax.xml.parsers, and org.w3c.dom libraries.

The Java.io library was used to handle and manage the inputs and be able to provide the output where the inputs and outputs were the messages and commands that were being exchanged by the server and the client. The Java.net library was used in order to create a socket for the purpose of the server and client being able to communicate to one another. The javax.xml.parsers and org.w3c.dom libraries were used so as to achieve and find the largest server in a given xml file.

The design and development of the code was divided in stages. First is the initial connection which is the handshake, second is receiving the job from the server, third is finding the largest server which is the (XML Parsing), fourth is scheduling each job to the largest server, then placing second through the fourth stages in a loop in order to loop each job and lastly is quitting and terminating the connection between client and server.

Everyone in the group was in charge of the handshake component; everyone managed to make the handshake work, Louise added a dynamic username authentication code. In the handshake component part of the code the socket was created first so that the client and the server are connected and can communicate to one another then a DataOutputStream and a BufferedReader that uses an InputStreamReader as its parameters were created through the socket and hence the client and server can now send messages to one another. The Client starts a conversation by sending a “HELO” command to the server the server replies with “OK”, the client then sends the message “AUTH ” with an authenticated username information the server replies with an “OK” message and an xml file containing the server information. The client then uses the server information to search for the largest server.

Inside the while loop is the place where the job scheduling component of the client is implemented. In the while loop, the client first sends the “REDY” command indicating that it is ready to receive a command, then the server sends a job, and the client receives the job and stores it in a string array called jobn where the array is split into different indexes storing different kinds of information about the job. Using the first index of the jobn array to create two if statements, one is to terminate the loop if the information stored in the first index is a “NONE” message and if it’s a “JOBN” message instead, the loop continues to code which schedules the job.

In this job dispatcher the jobs that were sent by the server to the client needs to be scheduled in the largest server. In finding the largest server the part was split with Randle in charge of trying the “GETS command” Seonghun and Louise are in charge of the “XML

Parsing", Louise managed to make her code work perfectly therefore, we used XML Parsing in finding the largest server. In order to find the largest server, the JDOM xml parser is used to search through the 'ds-system.xml' file that was produced by the server during the handshake. To find the largest server which is the server with the highest core count we accessed the server elements through *.getElementsByTagName('server')* and then using *.getAttribute("coreCount")* to find the server size then sorting to find the server with the largest coreCount and the storing its server name using *(.getAttribute('Type'))* in a variable

In the scheduling part of the source code, using the various arrays and variables that was stored in the xml parsing part of the code Louise was able to schedule each job, then the client sends the message:

"SCHD "+jobn[2]+" "+serverType+" "+"0"+"\n"

where jobn[2] is the JobID of the job , serverType is the server with the largest coreCount and "0" being the serverID, the server then sends "OK"

Once all the jobs have been scheduled the client now sends a "QUIT" message to now terminate the connection between the client and the server, the server sends "OK" and then exits. Lastly, closing all the socket, input and output streams.

References:

- Easterbrook, L., Park, S., Valerio, R., (2021) DS-Sim [Source Code]. https://github.com/44401213SeonghunPark/DS-Sim/blob/main/ds-sim-master/src/pre-compiled/MyClient.java
- Lee,Y.C., (2021) DS-Sim [Source Code]. https://github.com/distsys-MQ/ds-sim
- Lee,Y.C., Kim,Y.K, King, J., (2021) ds-sim: A Distributed Systems simulator User Guide
- Li C. (2009) XML Parsing, SAX/DOM. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_769
- Mkyong. (2021) How to read XML file in Java – (JDOM Parser), Mkyong.com [Online]. https://mkyong.com/java/how-to-read-xml-file-in-java-jdom-example/