UNIVERSIDAD PANAMERICANA

Facultad de Ingeniería

Final

Distributed Calculator

Alumnos:

Javier Iñaki Hernández Esquivel 0223287

Abstract—

In the first part of the project, a java program was made that works as a calculator, the point is that the calculator is divided with the functions it performs, not like any program with functions and that you call the function and that's it. Rather, a calculator will be created which can perform its tasks in a distributed manner, where there is a graphical interface, client, node and/or servers.
This same calculator is going to work in a distributed way, this to simulate a larger program that needs the same thing so that in case one of the clients or servers goes down, the system continues to work.
In the second part of the project, a mesh of nodes is going to be made, which are several nodes that are going to be communicating with the clients and with the servers, this is important because in case any node fails, we have the other nodes as auxiliaries and that they continue sending and receiving data so that the program does not stop working.
In the case of clients, they will connect to our network of nodes and will be sending messages to it, which are the operations entered in the calculator and also the recognition ID of the calculator, in this case for be able to identify who sent the message and they will be receiving responses from the node.
In the case of the server we have to receive data, and as long as they are not a result or an identification message of some connection, then they will be able to carry out the operation of the same, here depending on the sign received, "x" or " and" operation calling the microservices to perform.

Key Words*—*(Distributed, computer, calculator, sockets, servers, clients, nodes)

I. Resume

In the first part of the project, a java program was created that works as a calculator, the point is that the calculator is divided with the functions it performs, not like any program with functions and that you call the function and that's it. Otherwise, a calculator is going to be made which can perform its tasks in a distributed manner, where there is a graphical interface, client, node and/or servers.

This same calculator will work in a distributed way, this to simulate a larger program that needs the same so that in the event that one of the clients or servers falls, the system continues to work.

In the second part of the project, a mesh of nodes will be made, which are several nodes which are used in case one of the nodes fails and we have others in case of failure.

In the case of clients, they are going to connect to our mesh of nodes and they are going to be sending messages to it and receiving it and depending on the message and some conditions it is accepted or rejected.

In the case of the server, we have to receive data, and depending on the case, the operation performed will return or the server will remain the same without performing operations.

In the third part of the project, the connection will be made to all the nodes through our servers and clients, they will no longer connect only to one node and the others remain on hold, but all the nodes will be active connected to all my components.

II. Theoretical Framework

Distributed System

A distributed system is a set of computer programs that use other external resources or services on several different compute nodes to achieve a common shared goal. The purpose of distributed systems is to eliminate central failure points in a system. This is in case some of the external services are failing, so these systems can be recovered without problem and do not end their execution. In case of a non-distributed system, in case any of the systems fail, this makes the whole program stop working. For this reason, this type of computing is important and must be implemented in most systems so that they can be scalable and do not have problems in the future in this regard.

1. Key terms

- Monolithic architecture: It is a traditional model of software that is compiled as a unified entity and that is autonomous and independent of other applications.
- Microservices: It is an architecture where applications are divided into their smallest and independent elements among themselves, which are independent elements that work together to carry out the required tasks. Each of those elements or processes is a microservice.
- Autonomy: Each component of whichever service is in a microservices architecture can be developed, deployed, operated, and scaled without affecting the performance of other executable services within the application. Services do not need to share any of their code or implementations with other services.
- Specialized: Each service is designed with a specific set of capabilities and is focused on solving a specific problem. This seeks to prevent extra code from being added and the service from becoming more complex over time. If necessary, different capacities can be divided into smaller services.
- Agility: Microservices make it easy to organize small, independent teams that can work on individual services. This shortens development cycle times.
- Flexible scaling: Each service is allowed to scale independently to meet the demand for that feature of the application that the service supports. This helps to match infrastructure needs, accurately cost, and maintain availability in the event a service experiences an increase in demand.

- Easy Implementation: Continuous integration and delivery are made easy, which also reduces the difficulty of testing new ideas and reverting them if something doesn't work.

Centralized System:
A centralized system is where all the calculations are done by a single computer in one location, everything is done on the same side without a problem. The main difference between a centralized system and a distributed system is the communication pattern between the nodes of the system. The state of a centralized system is contained within a central node that clients access via a custom method, the central node performs all operations and sends data to the node only back and forth. All nodes in a centralized system access the central node, which can overload and slow down the network. Centralized systems have a single point of failure. This is not the case for distributed systems.

How do distributed systems help?
Distributed systems often help improve system performance, reliability, and performance. Reliability is improved so that central failure points and bottlenecks are eliminated. The nodes in a distributed system provide a type of redundancy so that if one node were to fail, other nodes exist that can take over and fix the error generated earlier. System performance is improved as nodes can be easily scaled horizontally and vertically. If a system is subjected to extensive load, additional nodes can be added to help absorb the load. It is also possible to increase the capacity of a particular node to handle extensive load.

- Technological freedom: Microservices architectures are not pigeonholed into a "one size fits all" design. Developers have the freedom to choose the best tool to solve specific problems.
- Reusable code: Dividing the code into small, well-defined modules allows you to use functions for different purposes. A service intended for a certain function can be used as a building block for another feature.
- Resilience: The independence of the services increases the resistance of the application to errors. In a monolithic architecture, a failure in a single component can cause the entire application to fail. With microservices, if there is an error in the entire service, that service will not affect the rest of the services.
- High reliability: The implementation of changes, improvements or updates for a specific service is allowed without running or minimizing the risk of the entire application falling, affecting availability.

III. First part of the project

At the beginning of the project, 3 java applications were made, which were divided into node, server and client. These 3 java applications would make up a larger project that would be a distributed calculator, where the tasks of each application would be divided to perform some operation.
First we will describe what each application does
For the client, we have a simple calculator, a graphical interface, in which the user can type in an operation that he wants to perform and here he will see the operation that the user wants and also the result that he receives once all the logic occurs. The client will send the node the operation to perform as well as receive the result from the node.
For the server, it will perform the operation, once it receives the operation message, and it will return this result to our node.
In the node, it will be in charge of sending the data to all the components that are connected to it, what it receives will send it to all equally, this is called broadcast.
Now, for the data to be processed and for it to work in the best way, the node needs to receive a message from either a client or a server, however, this node will forward this message to all the components that are connected to it. Each component such as the client or the server will receive many messages since the node is what it does, send these messages, however, each client or server knows if the message it receives is processed or ignored, depending on the type of message. whatever, if it is a result or if it is an operation.

IV. Second part of the project

In the second part of the project, a mesh of nodes is going to be made, which are several nodes that are going to be communicating with the clients and with the servers, this is important because in case any node were to fail, so we have the other nodes as auxiliaries and that they continue sending and receiving data so that the program does not stop working. This is because this way we have nodes in case of failure and broadcasting is not stopped at any time.
In the case of clients, they are going to connect to our mesh of nodes and they are going to be sending the

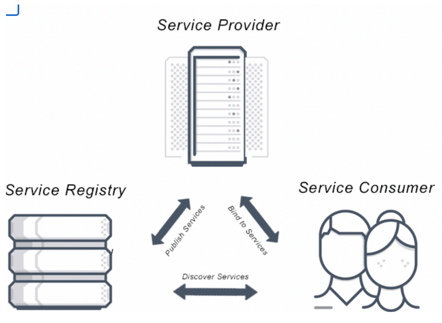
messages to it, which are the operations entered in the calculator and also the recognition ID of the calculator, in this case for to be able to identify who sent the message and they will be receiving responses from the node, and depending on the response, it is possible to show our result obtained in the calculator or not.
In the case of the server, we have to receive data, and as long as they are not a result or an identification message of some connection, then they will be able to carry out the operation of the same, here depending on the sign received, "x" or "y” operation calling the microservices to perform. The microservice is called separately, the server sends it the data it received and once it receives the response, it can send it to the node mesh without problem.

V. FINAL PART OF THE PROJECT

In this final part, what was done was that now we are going to connect to all the existing nodes within the node mesh. Before, the connection was made only to one node and there were the others in case one failed and it goes to the other, however, this can make it take time to search for another, so now all the components will be connected to all The nodes like this, there will be no problem that a node fails and a reconnection has to be made or something similar, the data is passed directly.
Similarly, we have the case that when there are less than three servers it is considered a risk and the client does not accept the result, we need three servers or more so that the system remains safe and without the risk of collapsing. In case 1 or 2 servers fail, we have 3 servers, of course this can be modified depending on the security you want, but in this project it was maintained with a minimum of 3 servers.
So, what happens if I have less than 3 servers, we are going to present an example where we have 3, and one of these servers fails, then the same system will have to duplicate one of these servers which have already failed, so that the account of servers are always maintained from 3 servers up and there is no failure.

VI. WHAT DO WE USED FOR THIS PROJECT?

The program will be carried out in the JAVA programming language.
We are going to use javaFx to start with to be able to make the graphical interface of the calculator, this works as a type of xml to be able to develop a frontend. We will then use libraries from java.io, which help us provide system input and output via data streams, serialization, and the file system. And finally java.net libraries, which allows connections and transactions to be made through the network. Using the java.net package we can communicate two or more computers that are in different parts of the world. In this case everything is local, it is only changing the ports etc.
We are going to make these javafx programs a jar, which are java executables which are used to only execute them from the terminal or any other side like any program you download to the computer.
Microservices were also used, which in java look like class loaders which in our code look like a jar.
The microservices, which are the components that distributed systems have, each microservice will be in charge of some specific task and it is easier to test this service separately. In case you want to modify, improve or fix this service for any existing reason, you just have to make this same service work and start to see how to modify it to reach your goal, it is not necessary to turn on other services, the same it is fixed and can be integrated into the system once what you want to do is solved.



VII. CLIENT SIDE

In the client side we can replicate it to it and thus we would have several separate clients running, sending and receiving data.
First, the client searches for the available nodes in the node mesh, searches between 5000 and 5200 and tries with each one until it can connect, then it sends the node that is a client, so that the node has a record of how many clients we have connected to the mesh.
The client will send the data to the node when the equal sign (=) is clicked on the calculator, what it does is send the data that is written in the calculator. We send the operation that are the numbers and the sign that we want to use separated by spaces to be able to read them without problem and carry them out. Similarly, what the client does is send its id, in this case the id was used as its unique port that this client has. This is so that the

Identify applicable funding agency here. If none, delete this text box.

node can know which client is who and that the message sent has a trace of where it is and where it returns.
The client will only show the message if that client has sent the message and also if there are 3 servers connected to the server mesh. It does this by means of what the node sends, it tells how many servers are connected and if it is able to display it.
For more information about the client code:
https://github.com/inaki321/calculadora-Computo-Distribuido/tree/master/Cliente

VIII. Server side

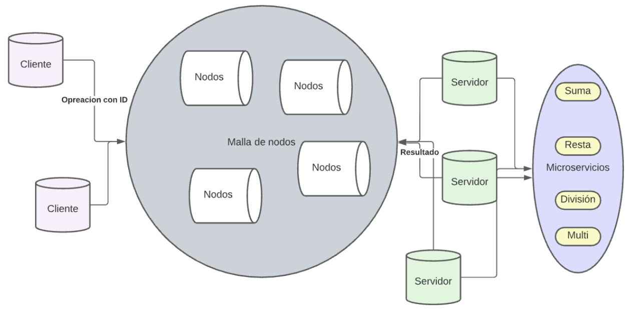
In the case of the server, when it initializes, it first searches for the available nodes in the node grid, it searches between 5000 and 5200 and tries each one until it can connect, then we send it to the node grid, which is a server that connected to so the node knows how many servers are connected.
Also, since we know that since we have a broadcast, then everyone receives messages sent by the mesh of nodes, so the servers will be receiving operations, and in the event that a server has a response, it would still send and receive it.
So what the server does is ignore any other message that is not an operation, if it is an operation it detects if it is addition, subtraction, multiplication or division, depending on the sign detected in the operation sent. Then, depending on the case, we send the operation to the necessary microservice, it generates an executable which performs the operation and once it is done, it returns it to the server. Once the server has the result of the operation, what it does is return it to the node mesh and it can return it with its respective client.
Now, on the part of the server, there must be 3 or more servers in the system so that it can work safely, this is because it is the measure that was implemented in this project, in case any of the 3 servers fail, there are 2 more left enough so that the program does not collapse. So, the server is duplicated in case there are less than 3 servers, to always have this limit.
For more information about the server code:
https://github.com/inaki321/calculadora-Computo-Distribuido/tree/master/servidor

IX. Microservices

The micro services are simple programs that are isolated and they work only when you call them. This is because you have to call an external executable to perform the operation and the program itself does not do it, this is so that the RAM memory does not overload and the systems have less problem in case the RAM is being overloaded either by a lot of data exchanged, an application that is failing, etc. So, our application is going to be lighter, the RAM is not going to overload because of simple operations such as sums, multiplications, divisions etc. This simple operation are going to be donde by the external executables.
For more information about the microservices code:
https://github.com/inaki321/calculadora-Computo-Distribuido/tree/master/microservice

X. Node

In the node we are going first to connect to a port which its number is between 5000 and 5200, in this case I wanted there to be 200 socket server type nodes between that range. Once the socket is created, we wait for the connections of the other devices that want to connect to our node.
When a device is connected, it has to check what type of connection it was in order to establish if it is a client, a server, etc. and to be able to keep track of their connections and what happens with the messages. With this, the nodes have a record of the clients, servers with their id to recognize them and how many they are. Just like when a connection shuts down and is disconnected from the server, its record that the node put at the time of connecting is deleted.
We are going to divide the mesh of nodes by node, what each node does is send messages in broadcast mode with the port from which I send the message to keep track of the messages that have been sent.
Similarly, the node with the registration of the servers, can notify the client if it can show its respective message that it would return or not, if the mesh has a count of three servers or more, then it notifies the client, if not, the message will be not shown on the client side.
To the existing nodes, all our clients and servers will connect without interruption unless the node crashes, then there is no problem that there is no communication between clients and servers, there is always continuous connection.
For more information about the node code:
https://github.com/inaki321/calculadora-Computo-Distribuido/tree/master/nodo



XI. Conclusions

The project was carried out in a good way, the objectives were met both in the first part and in the second part, since the broadcast was done in the correct way from the beginning, as well as each component carried out what was requested with the messages received, both the server and the clients knew what to do with each message and whether or not to ignore it. Similarly, it was possible to recognize messages among the entire system through the id, despite having a mesh of nodes in the data exchange in the system. The fact that the system works so that there are 3 or more servers was also achieved, the client will show result messages as long as there are 3 servers onwards, as well as the servers managed to do the operations through the microservices , which is to call an external executable to perform the operation and the program itself does not do it, this is so that the RAM memory does not overload and the systems have less problem in case the RAM is being overloaded either by a lot of data exchanged, an application that is failing, etc.

Also, in the end, it was possible to connect the clients and services to all the nodes so that there is no loss of time connecting to other nodes, but rather, they are already connected to each other and there is no loss of time, if any node were to fail, this is the others to continue doing the data exchange.

The only thing that was not achieved in the end, was to stop the "cancer" of the system, in case there are less than 3 servers, the system will start to duplicate some server so that we always have that range greater than or equal to 3 servers, without However, the cancer generated is not so serious either, since if we have 3 servers and 1 collapses, we will later have 2 servers plus other servers generated by n Nodes.

In other words, if we have 2 nodes and we have 2 servers, then we are going to have 4 servers, then the "cancer" of the system is not something serious, it is something controllable that does not cause the entire system to collapse, but rather it continues to function with a little of larger cells.

Now, this project was very good for realizing how important distributed computing is and since nowadays centralized programs cannot be made, since there are some systems that, if they were to fail, are more important to control this risk, it is not the same thing that a bullet train fails or that Facebook fails, in one you can lose lives and in another millions of dollars, however, all current systems must be distributed so that there is no type of loss and the system works without need from someone who is looking at how to repair it.

XII. References

1. https://clei2004.spc.org.pe/Peru/CS-USIL/Plan2021/2_12_Computacion_paralela_y.html#:~:text=La%20computaci%C3%B3n%20paralela%20y%20distribuida%20construye%20sobre%20cimientos%20en%20muchas,la%20memoria%2C%20y%20la%20latencia.
2. https://es.quora.com/Cu%C3%A1l-es-la-principal-diferencia-entre-un-sistema-distribuido-y-uno-paralelo
3. Victor Manuel Landassuri, Programación paralela y distribuida, UAEM, 2018, http://ri.uaemex.mx/bitstream/handle/20.500.11799/34870/secme-20264.pdf?sequence=1&isAllowed=y
4. Andrew S Tanebau, Marten van Steven(2001), Disrtibuted Systems Principles and paradigms
5. Raynal, Michel(2013), Distributed Algorithms for Message'Passing Systems
6. Random Code, 2017, Export Javafx 11, 15 or 17 projects into an executable jar file, www.youtube.com/watch?v=F8ahBtXkQzU
7. Deep Jain, Broadcasting and Multicasting in Java, October 2021, https://www.baeldung.com/java-broadcast-multicast
8. Phuoc-loc, 2013, https://www.google.com/search?q=java+broadcast+youtube&client=opera&hs=ekk&sxsrf=ALiCzsY868FBtJUeOw7g49cRXNxxrqu7Vg%3A1669254484169&ei=VM1-Y5CBCvTKkPIP47KEWA&ved=0ahUKEwiQ-8XE2cX7AhV0JUQIHWMZAQsQ4dUDCA4&uact=5&oq=java+broadcast+youtube&gs_lcp=Cgxnd3Mtd2l6LXNlcnAQAzIGCAAQFhAeMgYIABAWEB46BwgjELADECc6CggAEEcQ1gQQsAM6BggjECcQEzoHCAAQgAQQEzoICAAQFhAeEBM6BQghEKABOgQIIRAVSgQIQRgASgQIRhgAUIQDWJUJYPcJaAFwAHgAgAGmAYgB1QaSAQMxLjaYAQCgAQHIAQnAAQE&sclient=gws-wiz-serp#fpstate=ive&vld=cid:3b810cd0,vid:TNd6ktJl4ic
9.