Computo Distribuido

1st Santiago Valera Barreiros
Computo distribuido
Universidad Panamericana
CCDMX, México
0228654

***Abstract*—Nowadays the world depends on many information technology services, many of these services are hosted on servers but there are times when accidents happen such as power failure or a server burns down leaving millions of people without service, this is because most applications have their main program on a single server and not on several.**

I. Introduction

The use of electronic devices, especially those whose architecture allows them to handle data to store, process, exchange and protect them so that these data are transformed into information to help solve tasks was something that changed the world forever, since calculations were made in a matter of seconds or minutes that under other circumstances and method would take much longer, With programs that are capable of solving thousands of calculations in a short period of time, as time progressed people became more interested in computers because of this there was an exponential advance in both hardware and software and was evolving the main components that were used in a computer in the beginning were used vacuum tubes to reach the transistors which were doubling the amount that could be put on a chip each year.

As the software became more complex, more important features were added, such as communication between computers, which gave way to the client-server architecture.

A brief explanation of client-server architecture:

In the client-server architecture there are multiple clients that connect to a single server to retrieve the necessary resources to function, in this sense, the client is only a layer to represent the data and actions are detonated to modify the state of the server, while the server is the one that does all the heavy lifting, i.e. is the center of operations where calculations and logic are performed to deliver a result.

Where the server must expose a mechanism that allows clients to connect, which is usually TCP/IP, this communication will allow a continuous and bidirectional communication, so that the client can send and receive data from the server and vice versa, the Client and Server are developed as two different applications, so that each can be developed independently, resulting in two separate applications, which can be built in different technologies, but always respecting the same communication protocol to establish communication.

This system exists because it seeks to centralize the information and the separation of responsibilities, the server will be the only entity that will have access to the data and will serve them only to the clients of which it trusts, and in this way, we protect the information and the logic behind the processing of the data, in addition, the server can simultaneously attend to several clients.

This architecture has some advantages and disadvantages. The advantages are that the server acts as a single source of truth, it offers security since the server is usually well protected, it allows us to implement the business logic separately from the client, and the disadvantages are that an excessive number of users may not be supported by the server, if the server crashes everything stops working.

This architecture became very famous and almost all applications and systems use it such as video game platform servers, social networks where the client would be the applications on smart devices and the servers would be located in some center etc. [1]

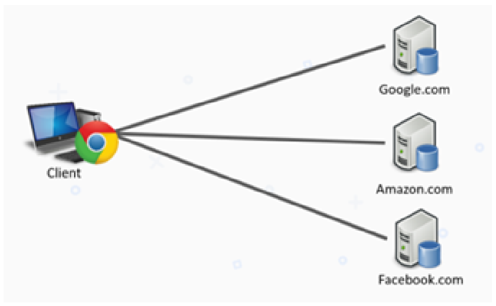


Fig. 1. Architecture Client-Server

Figure 1 refers to the client-server architecture where a client connects exclusively to a server and if the server is down the client is left without access to the server's resources, it can also be seen that a one-to-one connection is maintained with the server.

A. Network communication protocols

a) TCP (Transmission Control Protocol):: Is one of the fundamental protocols on the Internet, allowing applications to communicate with guarantees independently of the lower layers of the TCP/IP model. This means that the routers (network layer in the TCP/IP model) only have to send the segments (unit of measurement in TCP), without worrying about whether or not the data will arrive correctly. TCP supports multiple application layer protocols, such as HTTP (web), HTTPS (secure web). The TCP protocol allows flow control, i.e. it is able to mitigate the possible saturation of the network or the remote host. If a computer is transmitting at a speed of 500Mbps, and the destination computer can only receive information at 100Mbps, the TCP protocol adapts dynamically. In this way, the TCP protocol will always try to make the best use of the available bandwidth between source and destination. TCP also has congestion control, so that packets are not lost on the Internet because of router congestion. If the router is not able to process or forward packets at the rate at which it receives them, the router itself will discard them and they will be lost, as its buffer will fill up.

b) UDP protocol: The UDP protocol (User Datagram Protocol) is one of the fundamental protocols on the Internet and, like the previous one, allows applications to communicate with guarantees independently of the lower layers of the TCP/IP model. This means that routers (network layer in the TCP/IP model) only have to send datagrams (unit of measurement in UDP). UDP supports multiple application layer protocols, such as the popular DNS and even the DHCP protocol to obtain (and provide) IP addressing automatically. This protocol does not provide any type of flow control; if one device is faster than another and sends information, it is very possible that information will be lost because it will collapse the slower one, and we will have to resend the information. An important detail is that the management of datagram forwarding is carried out by the transport layer, since UDP is very simple and does not have any control mechanisms for datagram forwarding because it has been lost. UDP also does not provide any type of congestion control; if there is congestion in the network, packets could be lost and, logically, it will not be responsible for forwarding them as is the case with TCP. Therefore, since UDP does not have congestion control, flow control or error control, it could be said that UDP is an unreli- able protocol. Furthermore, it does not provide order in the datagrams sent, nor does it provide information on whether a datagram has arrived correctly, since there is no confirmation of delivery or reception. Any kind of guarantees for the transmission of information must be implemented at higher layers. [2]

B. PROBLEM TO BE SOLVED

Currently almost all applications and systems use client-server technology and one of the big problems is that if something happens to the server as a physical incident where the power goes out, the facilities are burned, a piece of hardware stops working which causes the server to stop working, if someone steals the equipment or if a virtual incident happens where a virus infects the server and left it inoperable, if it is a victim of a hacker that does not let it perform as it should or if the operating system has any failure, there is also a risk of bottleneck where the operations sent to the server are much larger than what this can deal with, exceeding its resource utilization can lower the efficiency of the server, if any of these things were to happen the server, so to speak, the server goes down this would cause all of its clients to be offline without the possibility of doing any other task, that is to say, if only one computer equipment goes down (which only applies to the server) the whole system goes down and the system operates normally when they change the server using a backup or repair the one that is unusable but this consumes time which is perceived by the users, we also have another type of problem when managing a server and it is the famous bottleneck that is not necessarily caused by the quality or age of the components, but by its performance. Bottlenecks are not limited only to high-end systems, since balancing is an equally important factor in systems with more basic hardware. A bottleneck refers to a component that limits the potential of the rest of the hardware, due to differences in the maximum capabilities of the two components. That is to say, when there are a large number of clients connecting to the same server, the server is not capable of dealing with all the clients, causing much slower responses from the server or, in the worst case, it simply stops working. For all these reasons, it is necessary to look for a solution that allows us to eliminate the dependence on a single server, improving the availability and efficiency of systems and applications that use clients and servers.

Another problem present in the servers is that of functionality, that is to say, a server has one or several tasks to perform, these tasks can fail and if the tasks are all united in one it means that if any of them has an error it can cause the whole system to stop working.

Another problem of everything being together would be to update a server only for one type of task, updating the task would mean stopping the functionality of the server completely which would leave the system without operating during the time that the task update is applied.

II. Definition of the project problem

A. Definition of the problem to be solved

The problem to be solved is to change the typical implementation of a client-server architecture that allows us to eliminate the problem of relying on a single server.

B. Hypothesis

An architecture that instead of relying on a single server and implements a series of nodes that are connected to each other, each with its corresponding cells, will allow an application to continue running even after a server goes down.

III. SOLUTION

The solution is clear and what we have to do is that we have to decentralize operations on a single server and do it in a distributed way so that if a server fails but we have more that are running the system can continue if you fail so you have to implement a distributed system that is a set of independent computers work as one in the eyes of the user, increasing the capacity and speed of processing and storage, so notorious.

Distributed systems are independent of the components that make them up, providing high reliability and guaranteeing high availability.. [3]

a) T: he main characteristics of distributed systems are:

- Concurrency: A distributed architecture allows it to be used by all users interacting in the network.
- Modularity: This characteristic allows distributed systems to be scalable, having the capacity to grow in a simple and efficient way.
- Transparency: Providing users and applications with a view of the system's resources as if it were a single machine or equipment.
- It does not depend on components: A distributed system does not depend on the different hardware components that make it up, since, if one fails, the others continue with the processes without the system being interrupted or suffering data loss.
- Openness: Distributed architecture allows the addition of new services that share existing resources without harming the services that are already running. For this reason, they must be designed on standard protocols that allow the use of hardware and software from different manufacturers or developers.
- Lack of global clock: The coordinations for the transfer of messages between the different equipment for the resolution of one or several tasks, do not have a general timing, that is to say, it is distributed to the components.

An implementation of this type brings with it many advantages such as: an increase in efficiency: since the task no longer falls on a single server and is spread over several, greater speed: having more servers there are some that have less load and can respond faster, greater tolerance to errors: if a server falls nothing happens because there are others that are already online and support it, flexibility and scalability: you can more easily increase the number of customers because there is a network that has several servers, in turn this network supports more servers are raised.

One would think that where there are advantages there are also disadvantages and the reality is that this type of system only carries a disadvantage and it is not the system itself but the person as it increases the complexity of implementing such a system, it is not as simple as a client server architecture, so the disadvantage only comes if the person in charge of the system does not have much experience and is not as skilled in handling this technology.

A. *SOA TECHNOLOGY*

Service-oriented architecture (SOA) is a type of software design that allows the reuse of its elements thanks to service interfaces that communicate over a network using a common language.. [4]

A service is a self-contained unit of one or more software functions designed to perform a specific task, such as retrieving certain information or executing an operation.

It contains the data and code integrations needed to perform a complete and distinct business function. It can be accessed remotely and interacted with or updated independently.

In other words, SOA integrates software elements that are implemented and maintained separately, and allows them to communicate with each other and work together to form software applications in different systems.

Advantages of SOA architecture include

- Easier use of infrastructure:we allow developers to take the functions of a platform or environment and adjust and implement them in new ones, a maintenance.
- Easier maintenance: since all services are autonomous and independent, each one can be modified and updated as needed, without affecting the others.
- Scalability: SOA makes it possible to run services in various programming languages, services and platforms, which increases scalability considerably. In addition, it uses a standardized communication protocol with which companies can reduce the interaction between clients and services, allowing applications to be tuned with less pressure and inconvenience.

B. *ADSOA*

The solution to the problem is to implement an architecture that does not depend on a single server and is able to recover if software components are missing. Eliminating the concept of a direct communication between the client and the server and adding an intermediary between these two in such a way that we can add several servers and if one of them fails another one of them will replace it. This intermediary is a data field which is a network of nodes. At the same time, within this solution, which is based on a decentralized architecture, SOA technology will be added in order to have an Autonomous Decentralized Service Oriented Architecture system.

a) Always connected components: In the solution an algorithm has been implemented that allows the components to be always connected, both clients and servers. Generally in this project there are cells which can be clients or servers and these are connected by an intermediary, the node, so that if a node falls the cells are able to search for another node within the network to avoid being without connection, this same network of nodes allows access to the client cells to all servers connected to the nodes so that they do not depend on only one. This network of nodes is known as the data field and is the component between the cells.

b) Self-recovery: Some client services need several servers to operate, but what happens if there are not enough servers, the system stops working? If the client detects that there are not all the necessary active servers to solve its request, it will ask one of the servers that responds to it to clone itself, so that its request can be processed, what happens is that the server accepts this request of cloning and raises another program similar to it, that helps it to solve the tasks, this way the system is able to recover without human intervention since it is able to detect when it is missing components.

c) Cancer: Although the cloning of a program can solve the problem of lack of servers, it can also cause a problem in that if too many programs are cloned, it can end up consuming all the resources of the machine, so it is necessary to clone only the programs that are necessary for the system to operate.

C. OBJECTIVES

Implement a functional calculator.

Make the calculator work even if not all servers are up and running.

Create a functional distributed architecture

D. SCOPE

This data field is a set of nodes that are connected to each other, which forms a mesh of nodes and the cells are connected to the data field, in this way an architecture has been created where a client can access different servers and is no longer dependent on a single one, although the client still depends on the node to which it is connected.

E. METHODS

To communicate and receive information from different applications and from different addresses, the Java sockets class will be used where the Socket class will be used to generate a client socket which is the one that will send the information. The client socket is generated by passing to the constructor the IP address to which it is going to send the information and the port on which the one that is going to receive the information is listening.

For the listening socket we will use the ServerSocket class whose constructor only requires the port on which it will be listening.

There are three ways to send the communication

- Unicast: where the signal sent knows who is the sender and who is the receiver.
- Multicast: where it is known who is the sender and who are the receivers.
- Broadcast: where it is not known who the servers are because the information will be sent to all available servers.. [5]

In the implementation of this program a broadcast type transmission will be used since the information will be sent to all the addresses that have information in such a way that the concept of receiver is eliminated because it is not known to which specific port the information should arrive and it is simply sent everywhere.

To avoid the complete shutdown of a server due to the failure of one of its functions, the SOA architecture will be implemented, which allows us to create microservices on the server so that if a microservice fails the other functions can continue to operate, this architecture also allows us to update a microservice without having to stop the server or the other microservices as it is in a separate file and can be carried out separately or not together functionality tests.

F. CODE

The calculator project has 4 executable programs and for them to run there must be at least one calculator, a server, a controller cell and the node must be raised necessarily if not, there will be no connection between the cells, also keep in mind that for the program to run the software that must first run is at least one node, it is recommended to raise several nodes to raise several cells later.

The same program can be executed several times in different terminals which will create programs with independent configurations, for example if we decide to run 2 client calculators we will create 2 calculators that listen to messages in different sockets of the Server and send information to different nodes, this gives us the possibility to create many different programs running the same instances. Brief description of the programs that are in the project.

Client calculator: it is a simple program that has a graphical interface to enter operations.

Node: is a program that listens and sends information, which can be visualized through a simple interface, this is the means of communication of all the cells.

Server: it is a program that is in charge of carrying out the operations indicated by the calculators and returns a result, this program also has a simple interface that shows us which are the numbers that it receives and with which it is going to operate and later it shows the results of the operation.

Controller: this program is in charge of controlling the minimum acknowledgements that each operation must receive and can modify the microservices of the nodes that are connected to the network.

G. Client Calculator

The calculator has a graphical interface whose class is controlled by HelloControlle and has two threads, one that is in charge of receiving messages and another one to send them. As global variables we have VB which is a verticalbox, sc which is a scrollpane, screen which is a label and finally a global variable called package of the class Package.

Package class description: the package class stores the message that you want to operate and the message that you want to display, the object that is instantiated from this class will be sent to both clients and servers so to know if the message corresponds to a server or a client a code nomenclature was established, if the message has a code 0 means that the message is only useful to servers, if it has a code 1 the message is useful to clients.

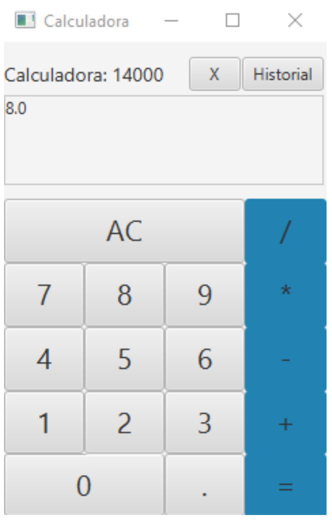


Fig. 2. Architecture Client-Server

Fig. 2 is the graphical interface of the calculator, it should be noted that each button has a functionality and is not simply aesthetic, the top of the calculator says node and then a number which indicates which port the server socket has been created.

When each calculator is started, a fingerprint is created by means of a hash to which is entered the moment in which the calculator was created, that is to say the time with the information of the month, day, hour, minute, second and the port in which the calculator is listening, with this fingerprint is intended to identify to whom a message is identified, so if an operation leaves a calculator we make sure that it arrives to that same one and not to another calculator. The buttons from 0 to 9 and the operations button modify the string attribute of the package variable adding as content the pressed element and then they call the method updateScreen() that updates the value of the label screen with the pressed element. The buttons that show the operators also update the value of the string and also when pressed indicate what type of content code it will have, which will affect the number of servers needed to display a result.

The button x allows us to see a table with the events that are stored for lack of servers and so that this information is shown it is sent to call a new screen that will be a class of pop up that is called TableView.fxml of which its controller will be obtained in order to be able to send the information that needs and this is displayed, the information that provides us is the following one. It shows the footprint of the event, in which round of sending messages to the node is in which it goes and finally it shows us which are the acknowledgements that it has received and how many times each one of these has arrived, it is necessary to emphasize that the information of this event will be visualized until the event in question has been processed by the servers that it requires.

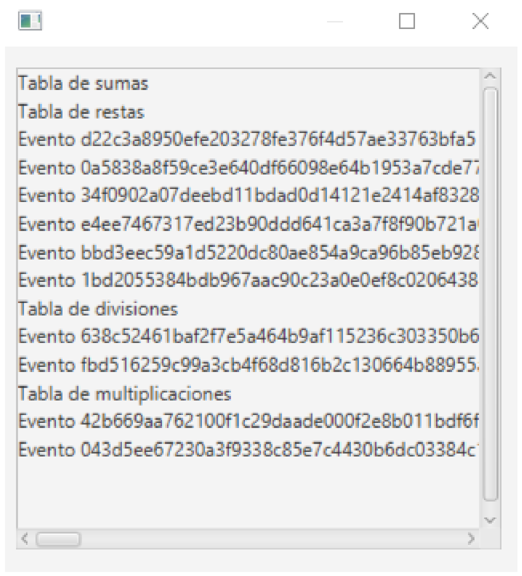
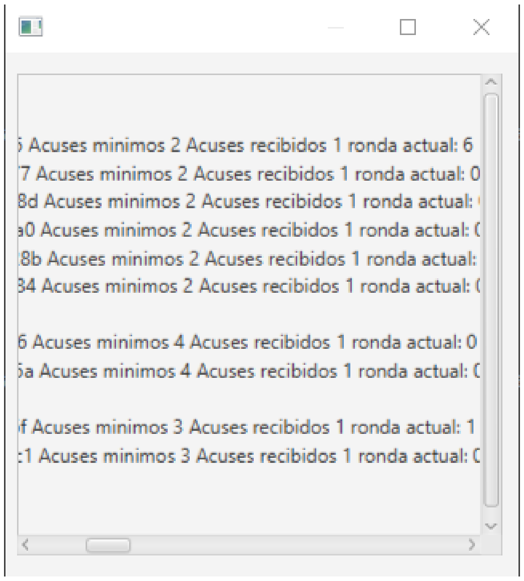


Tabla de sumas
Tabla de restas
Evento d22c3a8950efe203278fe376f4d57ae33763bfa5
Evento 0a5838a8f59ce3e640df66098e64b1953a7cde7
Evento 34f0902a07deebd11bdad0d14121e2414af8328
Evento e4ee7467317ed23b90ddd641ca3a7f8f90b721a
Evento bbd3eec59a1d5220dc80ae854a9ca96b85eb92
Evento 1bd2055384bdb967aac90c23a0e0ef8c0206438
Tabla de divisiones
Evento 638c52461baf2f7e5a464b9af115236c303350b6
Evento fbd516259c99a3cb4f68d816b2c130664b88955
Tabla de multiplicaciones
Evento 42b669aa762100f1c29daade000f2e8b011bdf6f
Evento 043d5ee67230a3f9338c85e7c4430b6dc03384c

Fig. 3. Events table

Image 3 shows what appears after pressing the ”x” button, what you can see is a table of the events that have been performed in the calculator but could not be processed, inside the table you can see the hash of the event.


5 Acuses minimos 2 Acuses recibidos 1 ronda actual: 6
7 Acuses minimos 2 Acuses recibidos 1 ronda actual: 0
8d Acuses minimos 2 Acuses recibidos 1 ronda actual:
a0 Acuses minimos 2 Acuses recibidos 1 ronda actual:
8b Acuses minimos 2 Acuses recibidos 1 ronda actual:
34 Acuses minimos 2 Acuses recibidos 1 ronda actual:

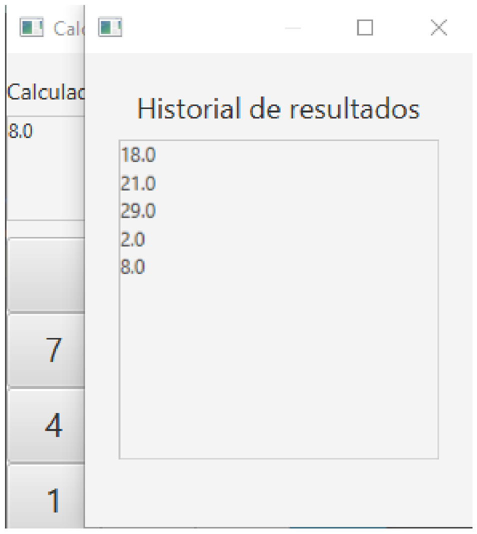
6 Acuses minimos 4 Acuses recibidos 1 ronda actual: 0
5a Acuses minimos 4 Acuses recibidos 1 ronda actual:

f Acuses minimos 3 Acuses recibidos 1 ronda actual: 1
:1 Acuses minimos 3 Acuses recibidos 1 ronda actual:

Fig. 4. Events table

La imagen 4 es la continuación de la imagen 3 y muestra cuales son los acuses mínimos del eventos, cuantos ha recibido y cuantas veces se ha enviado el evento hacia el nodo.

The button history shows us all the results that have arrived to the calculator which have been processed successfully for this a process identical to that of the button x is made where it is sent to call a new screen as a pop up which is called HistorialView.fxml, its controller is invoked and the necessary information is passed to it so that it can only show the results of the events.



Historial de resultados

18.0
21.0
29.0
2.0
8.0

Fig. 5. Log

Figure 5 shows a pop-up window that displays the log of operations that have been successfully processed and have reached the calculator.

When the send button is pressed, which would be the button with the equals symbol, a client socket is created that sends information of the message with the code assigned at the moment of assigning an operation, at the same time an event is generated which is stored in a list depending on the type of operation that has been assigned.

The special method initialize(). In this method the first label is added and then thread1 is initialized. In this thread a SercerSocket will be placed that will be listening for new messages, the messages will be accepted inside a while loop and when a message arrives it will be analyzed if the code it has is 1, if so the message will be shown in the chat screen, if not, no other action will be performed since the message was not destined for the client. It should be noted that if you run the executable several times on the same computer is able to create a new calculator on a different port which is indicated at the top, where it says Node: port where port indicates which port is the calculator. History functionality, a button called history was added which if clicked shows all the messages that have arrived to the calculator.

The showLists functionality allows us to see the events that have been sent to the node but have not received enough acknowledgements to be processed. Also within the client program there is another thread of the TransmitterContinuous class, this thread is responsible for sending messages constantly to the node, these messages are those that have not received the necessary acknowledgements to be processed, this algorithm accesses a file that is provided by the main flow of the program which is seeing which are the events that have already been processed successfully and which are not, These last events are saved in the file which will be read by the second thread since it contains the events that need to be sent again, it is also important to say that only the first element of each array will be sent, that is, if an array contains 3 sum type events, only one sum will be sent and until the first element has been processed, another different event will be sent.

H. Node

This program can be run several times and will generate different instances, that is to say, each time it is executed in a terminal it will generate a program totally independent from the previous execution, with different server sockets and sockets in each program, besides, when it is initialized it will generate a graphic interface that indicates which port it is listening to.

Main tasks of the node.

The program starts in the initialize() function which contains a thread that in general has 2 tasks, the first one is to generate a server socket different from the other code executions and its other task is to receive messages from cells and nodes and forward them to cells and nodes as appropriate.

Algorithm

The code has two arrays, one of cells and one of nodes, these arrays will be filled as cells and nodes are connected to the network. Once the port on which the node will be listening for messages has been generated, it will start receiving packets.

To be able to connect with other cells and nodes, it has two if statements that analyze if a message wants to establish a connection with the node and what type of component is to be connected, a node or a cell, and depending on the type of component to be added, the element will be added to its corresponding arrays. If the message that arrives to the node is not a message that seeks to establish a connection then there is a series of steps in charge of sending cells to all its components so that the messages are not recycled.

First it is analyzed if the message that arrives is from a cell, if this is the case the message that comes from a cell will first be forwarded to all the cells that have connection with the current node and then the message will be forwarded to all the nodes with which the current node has connection, but before sending the message to the nodes it will be indicated in the packet that the message that is sent comes from a node, in this way it will be avoided that the messages are recycled. Now we will analyze the case in which a message is received from a node, since this message comes from a node, it will only be sent to its cells and not to the rest of the nodes to avoid message recycling.

It should be noted that the node has the ability to reconnect components, i.e. if a cell or node had already connected to this node previously and for some reason lost connection that component can be connected again to the same node and this node recognizes that it is reconnecting a component with which it had already established connection.

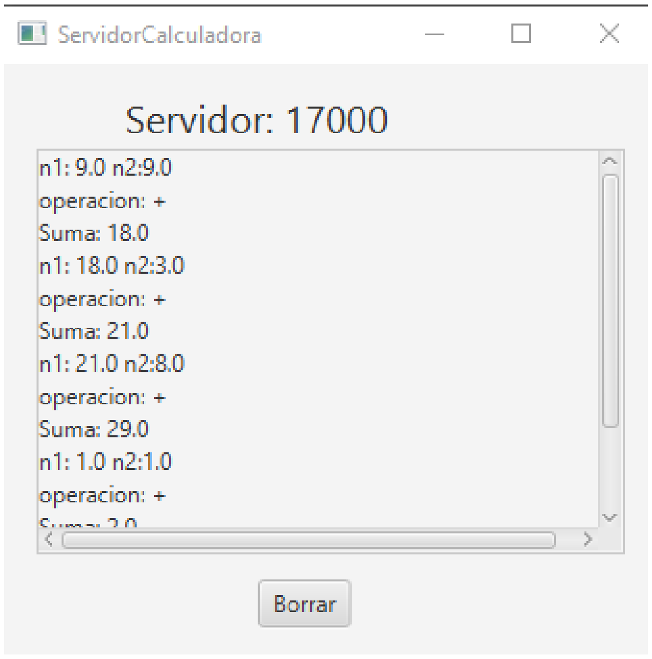
Fig. 6. Architecture Client-Server

Fig. 6 shows the node's graphical interface and at the top is the port used by serversocket to listen to messages.

I. Server

The main consists of the instance of a single thread whose class is called Process1, the operation of this is that it has a serversocket that is listening for messages, within a while cycle it will begin to receive the messages that arrive, once accepted it will see what type of code it has, if code between 1 and 4 0 the message is accepted and it will be worked with, otherwise the message will be discarded, When a message is accepted, the function calculate is called in which the incoming message will be operated and the result of the operation will be assigned in the string attribute of the Packet class and also the code related to the client will be assigned to the message, so that the client accepts it, once the operation is done the message is sent back to middleware.

It should be noted that on the server side the SOA architecture has been implemented more specifically is the implementation in the method calculate, within this method is a switch where depending on the type of operation will perform an operation by calling the method loadDynamic which is passed by parameters the function to be done, the class to be searched and the two numbers to operate. cargaDinamica: within this method the microservices are implemented, first a .jar file is searched where the code that we are going to use is found, this file is transformed into a URL so that the class can be loaded with the URLClassLoader constructor of the ClassLoader class which allows us to obtain the classes from the . jar file, then with the method ,loadClass() we obtain the class, the one that is passed by parameters, then the methods of this class are obtained and with .getMethod() we obtain the method we are looking for passing the name by parameters, finally this method is saved in a variable to which the invoke() method is going to be called, which will invoke the saved method and execute the procedure.e client, so that this one accepts it, once the operation is done the message is sent back to middleware.



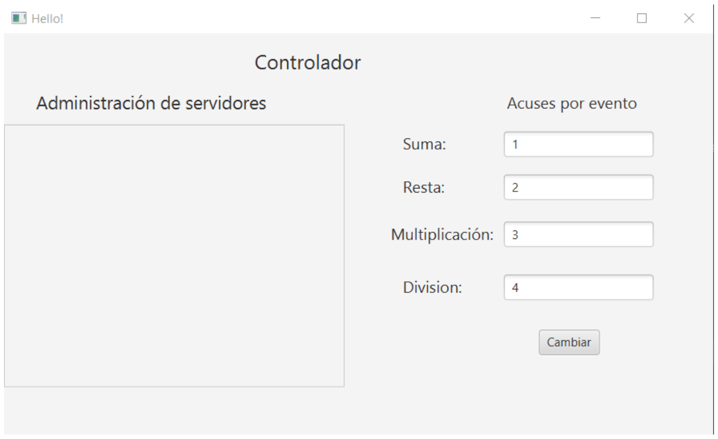
```
n1: 9.0 n2:9.0
operacion: +
Suma: 18.0
n1: 18.0 n2:3.0
operacion: +
Suma: 21.0
n1: 21.0 n2:8.0
operacion: +
Suma: 29.0
n1: 1.0 n2:1.0
operacion: +
Suma: 2.0
```

Fig. 7. Architecture Client-Server

Fig. 7 shows the graphical interface of the server, which receives messages to operate them and give a result, as can be seen in its interface that shows the numbers received, the operator and the result of the operation.

J. Controller

The controller program has a simple interface from which you can control the minimum number of acknowledgements needed for a calculator operation to be processed, from the input you can write the number of acknowledgements you need and when you press the change button a message is sent to the network of nodes which will reach all the cells, In this way the system is updated and for the microservices part the controller contains the microservices and by selecting a server cell and typing the microservice that you want to give the controller cell will give that specific microservice only to that cell. Each time a server is added to the network the node will send it to the controller in such a way that the controller will know the data of this server and will show it on the screen displaying the footprint of the server and this is how it is able to distinguish between the different servers.



Suma:	1
Resta:	2
Multiplicación:	3
Division:	4

Fig. 8. Controller

Fig. 8 shows the graphical interface of the controller , in the left is where are going to display the servers that are connected to the system and in the right are the entries where you can configure how many acknowledgements need each operation to work.

IV. CONCLUSIONS

In conclusion, the ADSOA architecture eliminates the problems of a client-server architecture, allowing us not to depend exclusively on a server, so that if one of the servers goes down, the system can continue working and in case another one is needed, the server can be raised again automatically without the intervention of any person. At the same time it is important to highlight the importance of the data field which is the means of communication between clients and servers where if a network node fails the cells can search for another node. This architecture should be implemented in critical systems since this system has the characteristic of self-recovery. It should be noted that although this system already solves many problems of the typical architecture used there is still a problem and it is in the nodes because if a node falls there is no way to recover automatically and if they are falling consecutively it could become a problem because eventually it would reach the situation that the system depends on a single node. Therefore, it is suggested to continue with the development of this project to submit it to tests in order to find more failures of this architecture and improve it.

REFERENCES

[1] Arquitectura Cliente-Servidor. (s. f.). Recuperado 27 de octubre de 2022, de https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/cliente-servidor

[2] Luz, S. de. (2022, 20 octubre). ¿Qué protocolo es mejor?: TCP vs UDP, descubre cuándo usar cada uno. RedesZone. https://www.redeszone.net/tutoriales/internet/tcp-udp-caracteristicas-uso-diferencias/

[3] Atlassian. (s. f.). ¿Qué es un sistema distribuido? Recuperado 27 de octubre de 2022, de https://www.atlassian.com/es/microservices/microservices-architecture/distributed-architecture

[4] ¿Qué es la arquitectura orientada a los servicios? (s. f.). Recuperado 27 de octubre de 2022, de https://www.redhat.com/es/topics/cloud-native-apps/what-is-service-oriented-architecture

[5] Martínez, C. Y. (2016, 28 marzo). Diferencias entre transmisiones Unicast, Multicast y Broadcast. Tajamar Tech Riders. https://techriders.tajamar.es/unicast-multicast-broadcast/