High Performance Computing

Authors: Bernardo Augusto, Marcelo Pereira and Daniel Marçal

Student number: 2814, 2691, 2675

Curricular Unit: High Performance Computing

Teachers: Norberto Albino and Laercio Júnior

Index

Introduction

High Performance Computing (HPC) provides researchers the ability to expand their data processing, simulation and computation across hundreds of cores (HPC @ QMUL, n.d.). In the last few years, there has been an exponential growth in number of HPC systems available for researchers and this has led to widespread use across many fields. Nowadays, parallel communications infrastructures, particularly computer clusters, have been crucial assets for industry and scientific communities considering its capability for handling high demanding tasks processes and performance-to-cost ratio (Costa, 2017).

A connection between two machines is like a hypothetical "cable" that runs between the two computers and each end of the cable connects into a socket. The socket API is an Interprocessing Communication (IPC) programming interface originally provided as part of the Berkeley UNIX operating system (The Socket API, n.d.). A process that "wants" to communicate with another process must create an instance or instantiate. Then the two processes send operations provided by the API in order to send and receive data.

The API sockets have some scientific applications. The program used in this project could easily be utilized for scientific purposes, with some adjustments. There will be some suggestions on what could be implemented in the program to make it a viable option for the mentioned field.

As for everything in life this interface is no exception. It has some advantages and some disadvantages. Even with its own disadvantages this remains a viable choice for most users.

API Sockets

A connection between two machines can be imagined like a hypothetical “cable” that runs between the two machines. And each end of the cable connects into a socket. The socket API is an Interprocessing Communication (IPC) programming interface originally provided as part of the Berkeley UNIX operating system (The Socket API, n.d.). A process that wishes to communicate with another must create an instance or instantiate. Then the two processes send operations provided by the API in order to send and receive data. A socket can make use of two protocols which are:

- TCP – Transmission Control Protocol
- UDP – User Datagram Protocol

These protocols play the same role, but they do it in a different way. TCP divides messages into packets and delivers them in the correct order. It also provides requesting retransmission of missing packets. In case of UDP, it does not provide assembly and retransmission requesting features. Basically, it passes packets along.

In this project we wanted to establish communication between two machines. For this we decided to use a virtual machine with the Operating System (OS), Ubuntu, in the VirtualBox.

VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2 (Oracle, n.d.).

The other machine created to communicate to the machine referred before was created using another tool, Amazon Web Services (AWS).

AWS (Amazon Web Services) is a comprehensive, evolving cloud computing platform provided by Amazon that includes a mixture of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offerings (Rouse, 2020).

For starters we begin this project by creating both Virtual Machines (VMs). In the VM of VirtualBox, we had interface and so we could get the Server and Client java files from

the browser. But in the other machine (the one from AWS), we did not had interface. In order to solve this problem, we decided to upload the files to a repository at GitHub of one of the collaborators of this project. The link for its GitHub is https://github.com/bernar0507/High-Performance-Computing. In order to get the files available in this repository, we had to use the code bellow (first box is to initialize Git in our machine, the second is to clone the files and repository in Git):

```
git init
```

```
git clone https://github.com/bernar0507/High-Performance-Computing.git
```

After we got the files in both machines, we moved to the communication part. We decided that the AWS VM would be the server and the VirtualBox VM would be the client. In this part we came across a problem. The problem was that we could connect the client to the server because we did not have a specific server port open. We solve this problem with the instructions bellow (figure 1).

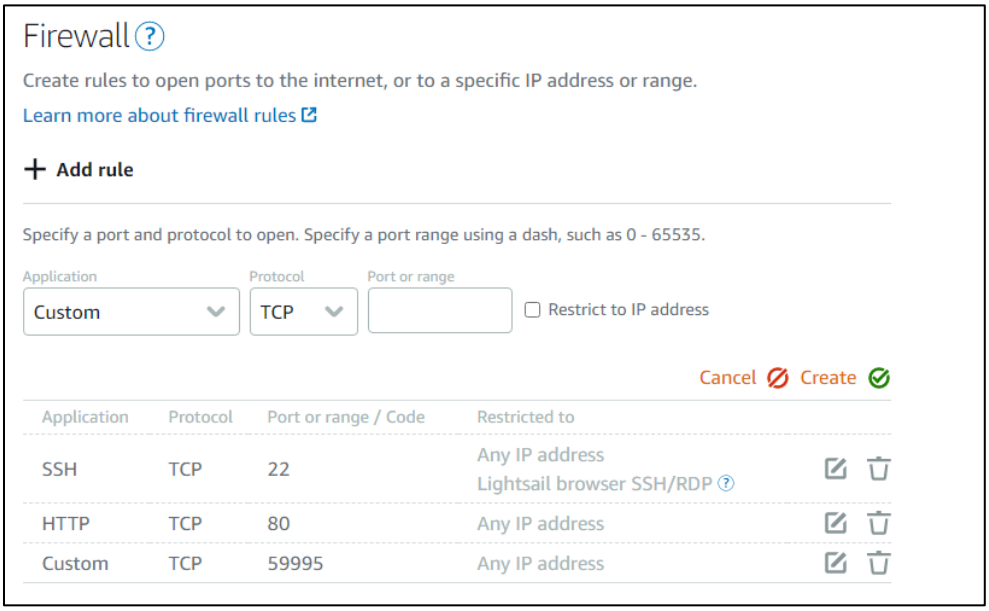

Figure 1 – How to create a new server port in order to our server and client could communicate to each other.

We had to insert a custom server port that followed the TCP protocol and the number of the port was 59995.

Next, in order to the client could communicate with the server we had to compile the files we had retrieve from GitHub. For this we have used the commands bellow (first on the client, and the other on the server).

```
javac Client.java
```

```
javac Server.java
```

After these commands we obtain two new files called "Server.class" and "Client.class".

Now that we have the class files in both machines, we could connect them. In order to establish the communication between these two machines we had to run a command. Available in the box bellow.

```
java Cliente ec-3-8-236-63.eu-west-2.compute.amazonaws.com
59995
```

The "ec-3-8-236-63" was obtained through the AWS interface. The section of the public IP address (figure 2) shows the number we referred before.

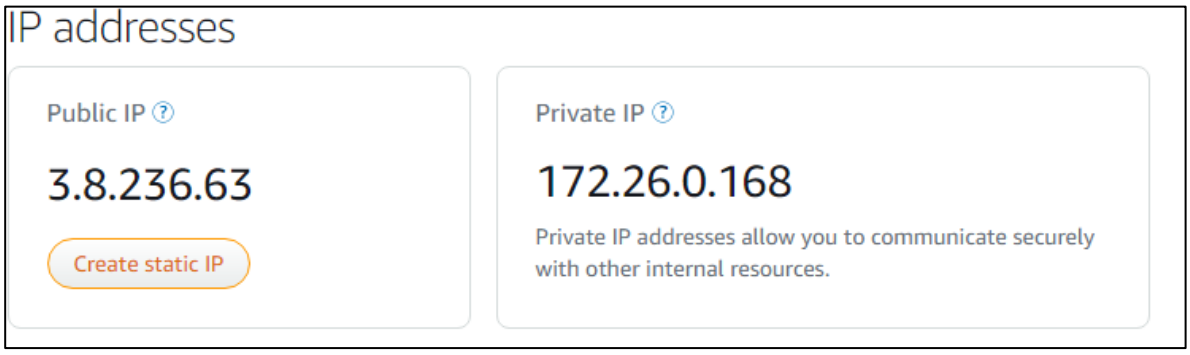

Figure 2 –Public IP that we need for the command above to work.

After we completed all these steps the communication was established, and we now could send and receive data. In the figure 3 and figure 4 we can observe the communication being used.

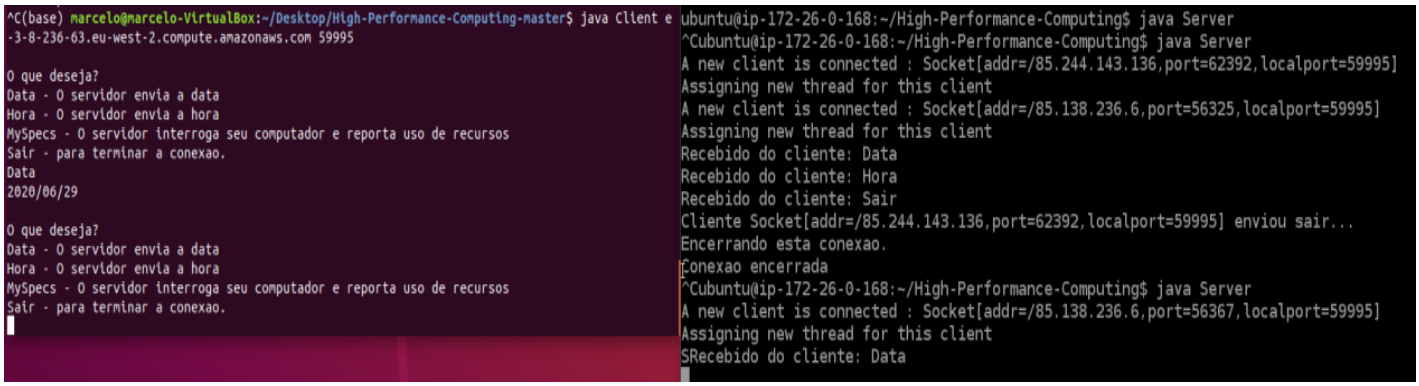

Figure 3 – In this figure we can see both (Server and Client) communication and the server questions

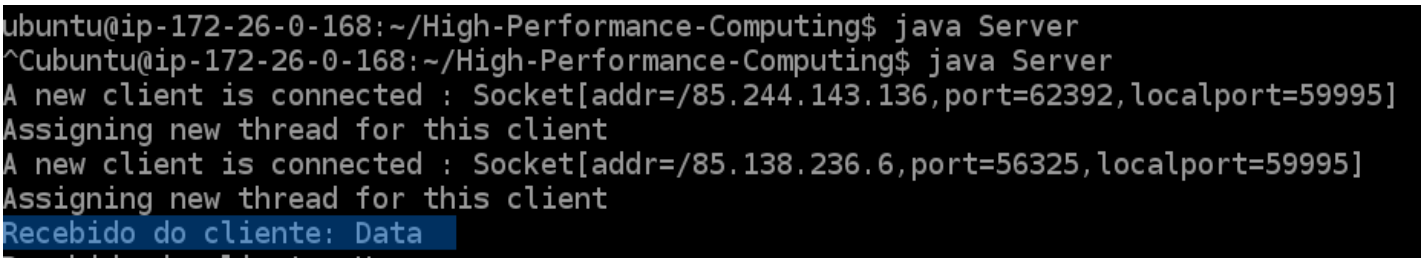

Figure 4 –Here we can observe the Server side with more detail and the demanded command from the "data" which the server responded with the current date

Scientific applications

The architecture used in this project was the client-server architecture. This type of architecture is powerful and can be used for other scientific purposes. A paper (Alexander Sturn, 2002) we have found demonstrates how this architecture can be used for high-performance gene expression data analysis.

Regarding this same subject, gene data analysis, it takes a huge advantage out of client-server environment which includes a powerful client for data preparation and results visualization, an application server for computation and an additional administration tool. The package is available free of charge for academic and non-profit institutions. (Alexander Sturn, 2002).

Another application is the communication between computers in a social network, where both user machines must synchronize both the IP and the port's number. In each conversation is established a point to be able to communicate. If you have 5 conversations at the same time, each one has its own number, but the IP address stays the same.

For the last one, we wanted to mention its impact on a multiplayer game's physics. The server needs to be the authority and they need to be calculated only once on the server. Then, depending on how the game plays out, we may need to do a client-side prediction as well. If one player moves their character at anything resembling a fast speed (basically, if the game is not turn-based), it will require a client-side prediction.

Advantages & Disadvantages

Companies and organizations seek opportunities to maintain the quality of its services and remain at the same or higher level of the competition. Client-server computing architecture in a company can increase tremendously its productivity.

The architecture Server-Client was the architecture that we have used in this project. It has shown that it can handle very well the job but beside its advantages, it also has some disadvantages.

Starting with the advantages, it has 6 mains advantages, these are:

- Improved Data Sharing
- Integration of Services
- Shared Resources Amongst Different Platforms
- Data Processing Capability Despite the Location
- Easy Maintenance
- Security

This architecture improves data sharing because the data is hold by usual business processes and manipulated on a server. In terms of integration services, each client receives an opportunity to access corporate information via desktop interface. This way the necessity of log into a terminal mode is eliminated. In the 4th point of advantages, the application used for client-server model is build regardless the hardware. So, it provides an open computing environment. In terms of data processing capacity, the client-server user can log into the system, not being dependent on the location or technology of the processors. It is easy to maintain because its easy to replace, repair, upgrade and relocate the server while the client is not affected. It also offers security, since servers have a better control on the access and resources. This way only authorized clients can access or manipulate data.

As mentioned at the start of this section, the client-server architecture also has some disadvantages. In terms of disadvantages, the client-server architecture lacks in the terms of following examples:

- Overload of the Server
- Centralized Architecture

In terms of overload of the server, it can happen when the server receives multiple client requests at the same time, this can lead to overload of the server. The next disadvantage is related to the fact that this architecture is centralized. This way if the server fails then the client request is not complete. This show that the server-client architecture should be more robust in order to be called a good network.

Conclusion

High Performance Computing most generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business (inside HPC, n.d.).

A connection between two machines can be imagined like a hypothetical “cable” that runs between the two machines. And each end of the cable connects into a socket. The socket API is an Interprocessing Communication (IPC) programming interface originally provided as part of the Berkeley UNIX operating system (The Socket API, n.d.).

The protocol used to establish the communication between the two machines was the TCP protocol.

With some adjustments to the program used in this project this could easily be used for scientific purposes, like physics, gene expression data analysis and social networks.

Besides the advantages shown for the client-server architecture it also lacks in some parts like the possibility of server overload and its centralized architecture. If the client-server could be “fixed” in terms of the centralized architecture, it could be called a good network.

Acknowledgment

This paper was developed by a group of three elements, Bernardo Augusto, Daniel Marçal, and Marcelo Pereira and with the supervision of the professors, Norberto Albino and Laercio Júnior and carried out at Polytechnic Institute of Setubal, Portugal.

References

Advantages and Disadvantages of Client-Server Architecture. (n.d.). Retrieved from https://sites.google.com/site/clientserverarchitecture/advantages-of-client-server-architecture

Alexander Sturn, B. M. (2002, December 16). BIOINFORMATICS APPLICATIONS NOTE. *Client–Server environment for high-performance*.

Augusto, B. (2020). Retrieved from GitHub: https://github.com/bernar0507/High-Performance-Computing

Costa, A. (2017, November). SooMPI - Socket over MPI.

HPC @ QMUL. (n.d.). Retrieved from https://docs.hpc.qmul.ac.uk/intro/

inside HPC. (n.d.). Retrieved from https://insidehpc.com/hpc-basic-training/what-is-hpc/

Oracle. (n.d.). Retrieved from VirtualBox: https://www.virtualbox.org/

Rouse, M. (2020, April). *TechTarget*. Retrieved from searchAWS: https://searchaws.techtarget.com/definition/Amazon-Web-Services

The Socket API. (n.d.). Retrieved from http://csis.pace.edu/~marchese/CS865/Lectures/Liu4/sockets