Virtualizing a Cluster to Optimize the Problems of High Scientific Complexity within an Organization

Enrique Lee Huamaní[1], Patricia Condori[2], Avid Roman-Gonzalez[3]
Image Processing Research Laboratory (INTI-Lab)
Universidad de Ciencias y Humanidades
Lima, Perú

***Abstract*—The Image Processing Research Laboratory (INTI-Lab) of the Universidad de Ciencias y Humanidades has several research projects related to computer science needing high computational resources. Some of these projects are associated with climate prediction, molecule modeling, physical simulations, and others these applications generate a significant amount of data, regarding the big data issue, despite having excellent hardware features, the final result is obtained after hours or days of calculation depending on the algorithm complexity. For this reason, it is not possible to present optimal solutions at an ideal time. .In this work, we propose the virtualization and configuration of a high-performance cluster (HPC) known commercially as a "supercomputer" that is composed of several computers connected to a high-speed network to behave like a single computer. The virtualization is used to run a scientific algorithm that will apply performance tests using four virtual computers to demonstrate that the reduction of time is achieved by using more machines and thus be able to be implemented in the laboratories of the institution.**

Keywords*—High-performance cluster; distributed programming; computational parallelism; supercomputer; high-efficiency computing

I. Introduction

High-performance clusters (HPC) or also considered 'supercomputers' are potent computers that perform calculation tasks at high speeds compared to an ordinary computer [1]. These clusters are used in digital processing for scientific research, big data, data mining, bioinformatics, remote sensing, image processing, medical imaging, stage reconstruction, realistic simulations for computational chemistry, etc. [2] as new and emerging scientific findings, it is necessary to use the maximum performance of a computer so that these can give optimal results in an ideal time. By this necessity, it is possible to implement the HPC architectures where they use the Central Processing Unit (CPU) to obtain floating point operations per second (FLOPS) that are the processing capacity of a computer [3]. The use of several low cost machines that are interconnected through a network to have a unique behavior began with National Aeronautics and Space Administration (NASA) in 1994, where they used recycled computers for the creation of a supercomputer, this was called the Beowulf project which was realized in the Center for the Excellence in Data (CESDIS) [4]. The idea of the construction of low-resource computing supercomputers was disseminated worldwide to scientific and academic communities so they decided to use their computing resources without the need to purchase assembled supercomputers due to the considerable costs that these generate by the specialized maintenance that can occur over time, making these unnecessary purchases in the future. The present work performs the virtualization of an HPC using four virtual machines to apply performance tests with a computational algorithm to prove its scalability and can be implemented in the institution. Due to the new proposal of projects, it is necessary to have equipment that uses the maximum computation resource; some universities use these architectures to carry out research. In Peru, there is the case of the Universidad Nacional de Ingeniería [5] that performs performance benchmarks to apply algorithms of high scientific complexity. Another university in Latin America is the Universidad de Quindío that uses HPC to carry out calculus of quantum-mechanical chemistry [6], there are also investigations related to urban traffic such as the case of [7] that performs simulations to allow researchers to address real-size traffic problems in large networks using powerful, precise approaches to network traffic. The use of this architecture is increasingly used worldwide as can be seen in the official page of supercomputers top 500, where it shows a list of the most powerful supercomputers in the world [8]. There is a significant increase in its implementation regarding the area of computer science; therefore, one must know the implementation, because it will be very required for the scientific community.

II. Methodology

High-performance cluster virtualization consists of 4 virtual machines that will have the same open-source operating system and process distribution package. Virtualization will be made up of 2 or more computers that are interconnected by a network computer to use the SSH protocol that facilitates secure communications between systems [9]. The machine that manages the algorithm and distributes the processes is called the master node, and those that receive the information to be processed in parallel are the slave nodes that will have a single purpose that is to give the result to the master node, Fig. 1 shows its architecture.

A. Master Node

It is in charge of administering and controlling the processes that will be sent to the receiving computers [3] its function is to distribute the tasks in equal parts to the desired amount of them that will use as a way of communication the Protocol Secure Shell (SSH). If one wants to see the ecosystem of the HPC graphical way, one can install monitoring

packages; therefore it is ideal that these have a graphical user interface, one can also access the public network to get updates of the operating system.

B. Slave Node

The computer that receives the algorithm and processes part of the problem is called the slave node. Its primary function is the processing of data designated by the master node [10] they are interconnected by a high-speed network where one has direct communication with the master node; it is recommended that the slave nodes do not have a graphical interface because they consume their computing resources. They do not need to be connected to the public network because their only function is communication with the master node it is recommended that all the computer that receives the algorithm and processes part of the problem is called the slave node. Its primary function is the processing of data designated by the master node [10]. They are interconnected by a high-speed network where one has direct communication with the master node. It is recommended that the slave nodes do not have a graphical interface because they consume their computing resources. They do not need to be connected to the public network because their only function is communication with the master node. It is recommended that all.

C. Communication Network

The interaction between the nodes is distributed through a communication network. In this virtualization is used a connector of Ethernet board thanks to this communication channel, the master node can spread the tasks to the slave nodes applying techniques of computational parallelism [11]. It is recommended that this network equipment does not include transit jobs outside the HPC operations because it can occur unbalanced at the time of obtaining results.

D. Multiple Data Multi-Instruction

The MIMD (multiple instructions, multiple data) is a technique that helps to achieve the parallelism between the nodes. Processors can run different instructions in different data [12]; it is recommended that this network equipment does not transit tasks outside the HPC operations because it can occur unbalanced at the time of obtaining results, in Fig. 2 its architecture is shown.

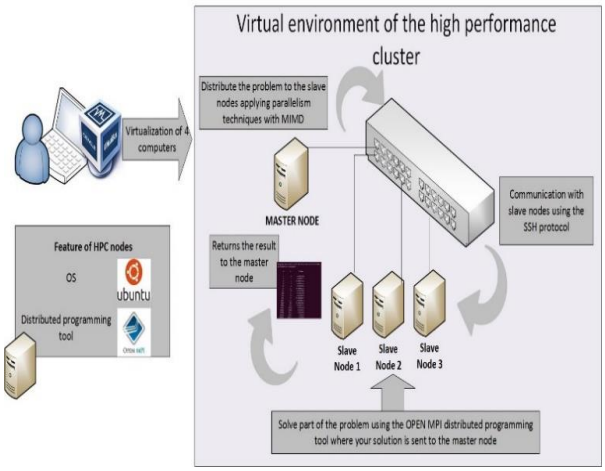

Fig. 1. Design of an HPC Architecture.

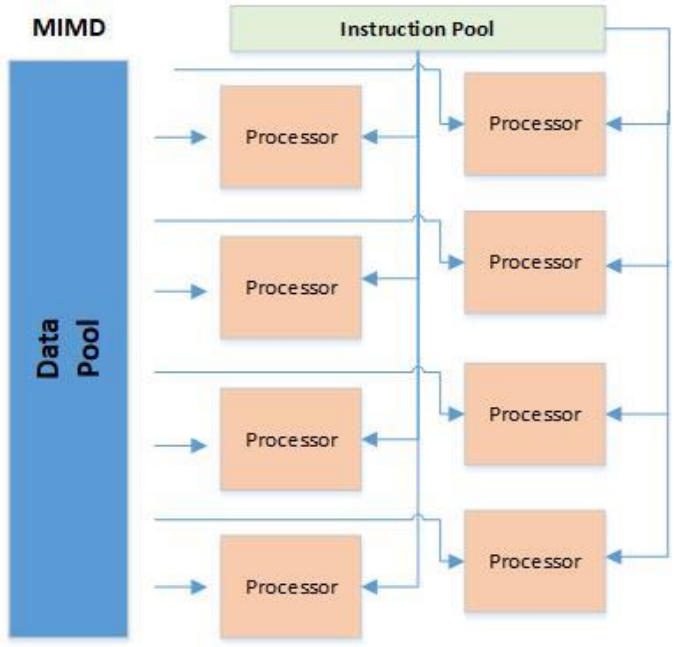

Fig. 2. Mingle Instruction Stream Multiple Data Stream (MIMD) Architecture.

E. Message Passing Interface

MPI (Message Passing Interface) is a specification for developers and users of message-passing libraries, mainly addressing the parallel message-passing programming model [13]. It is designed to be used in programs that exploit the existence of multiple processors. Different standards meet these techniques; among them, one has the MPICH, MVAPICH, and the Open MPI [14]. The tool used in this virtualization is Open MPI.

F. Virtualization

Virtualization is the most used in the world of computing, due to the advantage, it generates in saving energy, space, and management of the less physical machine. The virtualization tool used is Oracle VM Virtual Box, where four virtual machines are used, using one of them as a master node and the others as a slave node, as shown in Fig. 3.

Fig. 3. HPC Node Virtualization.

G. GNU/LINUX

All HPC nodes must have the same operating system that facilitates the interaction between the user and the project to be developed; an open-source operating system is used where modifications can be made without restrictions, in the Official supercomputer page [12] shows that most implementations are performed by open source operating systems, Fig. 4.

For the configuration, GNU/LINUX-Ubuntu was opted for having a friendly graphical interface and being open source, despite being one of the least used operating systems among HPC architectures, its scientific community compensates that one can always resort to having problems.

H. High-Performance Cluster Configurations

For the use of HPC, there is an open-source tool called Open MPI. It is an open-source message step implementation that is maintained and developed by a large consortium of academic partners, research and industry. (To access to download the package that contains the installer go to its official page www.open-mpi.org) [13]. It is important to download the most stable version in this case openmpi-4.0.1.tar.gz, then use the following command from the terminal to decompress the package: tar –xzvf openmpi-4.0.1.tar.gz.

After uncompressing specify installation point: ./configure –prefix=$HOME/openmpi then install: make all install and sudo apt-get install openmpi-bin, to conclude with: sudo apt install libopenmpi-dev.

When the installation is complete, the following is written to the terminal: export PATH=$PATH:$HOME/openmpi/bin and export LD_LIBRARY_PATH=$LD_LIBRARY_PATH :$HOME /openmpi/lib.

Then install the Secure Shell (SSH), this is a remote management protocol that allows you to launch commands and copy files from the master node to the slave nodes [14] with the command:sudo apt-get install ssh and the network file system is installed (NFS) which is the most used protocol for access to storage [15] with sudo apt-get install nfs-common portmap. From this point, the cloning of the nodes is started by assigning one of them as the master node.

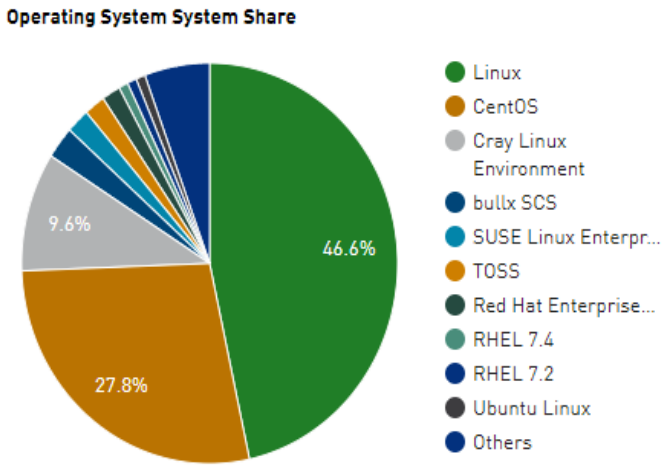

Fig. 4. Operating System Share.

From the master node the following command is entered: sudo apt-get install nfs-kernel-server, package that allows sharing the directory.

Each node has created a folder with the command: mkdir clusterdir, then a static IP is assigned with the same gateway, as shown in Table I.

An SSH key is generated from the master node where a copy is made to all the slave nodes in order not to ask for access at the time of processing; the following command is applied: ssh-keygen, where a unique key is generated as shown in Fig. 5.

Each of the slave nodes is accessed and a. SSH folder is created with: mkdir .ssh. The key is then copied from the master node to the slave nodes, as an example applies to the first node: scp .ssh/id_rsa.pub cluster-uch@172.16.9.201: Permissions are created. SSH with the command chmod 700 then from all the slave nodes the copying of the id rsa.pub is done: mv id_rsa.pub .ssh/authorized_keys. From the master node is accessed: sudo nano/etc/hosts, in it we add the static IPs of all the nodes of HPC with their respective prefix as explained in Fig. 6.

A modification is made to the export file with the command: sudo nano /etc/exports in it you enter the following:/home/cluster-uch/Clusterdir 172.16.0.0/24(rw,no_ subtree_check,async,no_root_squash.

TABLE. I. THE IP LIST USED IN THE CLUSTER

Name	IP	Gateway
Master node	172.16.9.200	172.16.9.254
Slave node 1	172.16.9.201	172.16.9.254
Slave node 2	172.16.9.202	172.16.9.254
Slave node 3	172.16.9.203	172.16.9.254

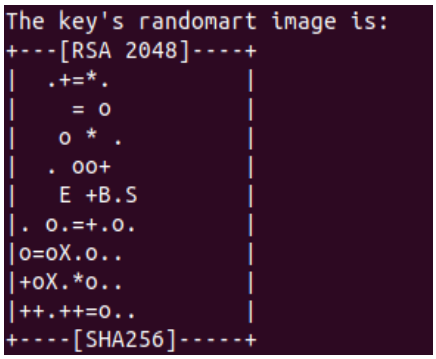

Fig. 5. Getting SSH Key.

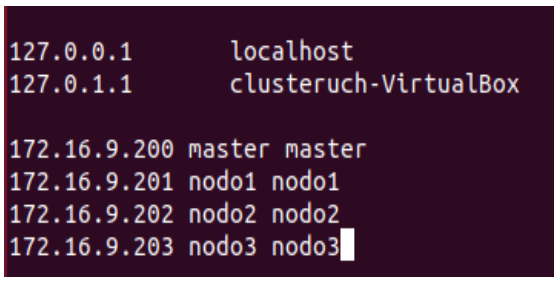

Fig. 6. Modification of the Hosts File.

The service is restarted with the command: /etc/init.d/nfs-kernel-server restart, rebooted the service applies the mount from each slave node: sudo mount–t nfs 172.16.9.200: /home/cluster-uch/clusterdir /home/cluster-uch/clusterdir.

As of last, a file is created from the master node with the following command: sudo nano .mpi_hostfile, and the number of nodes to be used by HPC is selected. As shown in Fig. 7, the Nodo1, Nodo2, and Nodo3 are used. In this case a kernel is used where it is assigned from the slots.

In most cases, the slave nodes do not have monitors, keyboards or graphical interface because they want to avoid that the computing resources are consumed by the graphical user interface which is eliminated with the following command in each one of the nodes: sudo apt-get remove xserver-xorg-Core.

A computational algorithm is used that is programmed with the C programming language where MPI applies. This algorithm calculates the time it takes to find prime numbers [16]. Fig. 8 shows the description of the algorithm.

```
#Master node

#localhost slots=1

#Slave node 1
nodo1 slots=1

#Slave node 2
nodo2 slots=1

#Slave node 3
#nodo3 slots=1
```

Fig. 7. Enabling Slave Nodes with an Exact Amount of Kernels to use.

PSEUDOCODIGO: sum of prime numbers

```
** Start of parallel calculation

Main structure () {
** Declaration of variables
Number i, id, n , n_factor, n_hi, n_lo, p, primes, primes_part, master;
Decimal wtime;
** Declaration of values
n_lo=1;   n_hi = 131072;   n_factor = 2;   master = 0;

** Initialization of MPI
Initialization _MPI();  p = amount_of_sloves ();  id = call_processes_range();

If   id is equal to master  Do {
Samples the header and the number of slave nodes P  to use}

**An initial value is assigned to make the journey
n = n_lo;
While n <=n_hi   Do {
If  id is equal to master  Do {
 ** The current process time is assigned   wtime = real_time_process()}
 ** Send a message from a source process to the group send_message_group (n,Int,master)
 ** You get a part of the problem    primes_part = prime_number(n,id,p)
 ** Reduce the problem from the root slaves Reduce_problem(primes_part, primes, master)
 If  id is equal to master  Do {
  ** Apply time reduction  wtime = real_time_process ()-wtime;
  Samples row of results  : n , primes, wtime; }
  ** Multiply the route by the factor  n =(n* nfactor);
 }
** Ends the MPI
Ending _MPI();**End of parallel calculation
}
```

Fig. 8. Parallel Code for Calculating Prime Numbers.

To execute the code we will access the directory Clusterdir where the following command is entered: mpic++ primos.c++ -o primos. This way you get a compiled file for your use.

III. Result

In this section, two performance tests are performed to make comparisons of scalability. The results of Table II are performed without the HPC architecture unlike Table III which uses 3 slave nodes and a master node, the final result shows three values where N is the number of processes performed, S the sum of the prime numbers and T the solution time, the following process is performed using the following command in its compiled directory: ./primos.

TABLE. II. Results with a Single Computer

N	S	T
1	0	0.000003693
2	1	0.0000001214
4	2	0.0000001077
8	4	0.0000001214
16	6	0.000000184
32	11	0.0000003319
64	18	0.0000007894
128	31	0.0000023744
256	54	0.0000095883
512	97	0.000282498
1024	172	0.003585893
2048	309	0.0260807
4096	564	0.0677598
8192	1028	0.196943
16384	1900	0.700947
36768	3512	0.700947
65536	6542	2.61356
131072	12251	10.1143

TABLE. III. Results with the HPC Architecture

N	S	T
1	0	0.0040071
2	1	0.000939131
4	2	0.00194287
8	4	0.00192809
16	6	0.00131488
32	11	0.00120115
64	18	0.000807047
128	31	0.000520945
256	54	0.000102179
512	97	0.000566006
1024	172	0.00029397
2048	309	0.000814915
4096	564	0.00234103
8192	1028	0.00741506
16384	1900	0.03195
36768	3512	0.101557
65536	6542	0.372828
131072	12251	1.33745

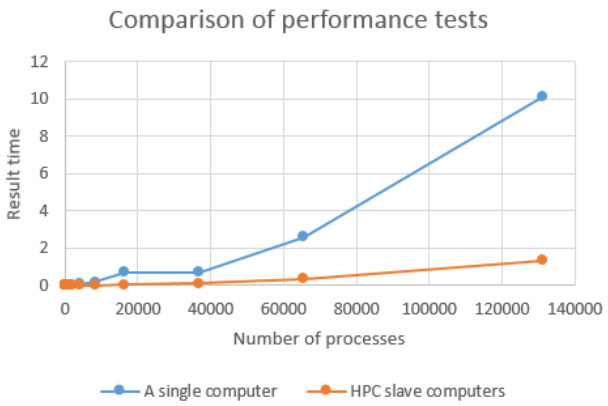

Fig. 9. Comparison of the Performance Test.

The algorithm is then executed using HPC virtualization within its Clusterdir folder with the following command: mpirun–np 3-hostfile ../.mpi_hostfile ./primos. The results are shown in Table III, where it is observed that the calculation of the last process takes 1.33745 seconds, which makes HPC virtualization meet the scalability and objectives defined.

Below is Fig. 9, where the comparison of an ordinary computer and the HPC virtualization is displayed, where it uses two axes that are the results time and the process numbers.

Fig. 9 shows a blue line that is the representation of a computer and the Orange Line of the HPC, as one can see, by using more number of slave nodes considerably reduces the time.

IV. Discussion and Conclusions

There are different ways to demonstrate scalability without resorting to virtualization, and one option would be the use of raspberry PI which is a low-cost computing platform, its configuration is similar to HPC virtualization in this case we have the work presented in [5] that performs benchmarks with its two raspberry PI cluster prototypes. It is always necessary to measure the number of FLOPs due to unbalance problem that may occur when a specific amount of slave nodes is used due to high-speed network bottlenecks. Some algorithms can give us FLOPs by using different amounts of nodes. A clear example is shown in [3] that use a package called Linpack that makes intensive use of the operations of floating per second applying basic linear algebra subroutines. These are applied by assigning different amounts of slave nodes when the FLOPs stop increasing; one must conclude that this is the ideal amount to use.

In this work, it is concluded that the virtualization of the cluster of high performance fulfills the reduction of time of the algorithmic processes thanks to the connection of computers that communicate using the protocol SSH. Therefore, this project performs as evidence for its implementation in the laboratories of the Universidad de Ciencias y Humanidades due to the results that show its scalability using the architecture to the comparison of a single computer. Also, it will contribute a benefit for the scientific community INTI-Lab that has thought in the accomplishment of machine learning applying techniques of Big Data using the architecture HPC that will use algorithms of high scientific complexity.

Reference

[1] G. Atul, G. Bhargavi, and K. Uditnarayan, “Study of Supercomputer ’ s Architecture , Application and Its Future Use,” p. 2, 2014.

[2] I. Ocampo and L. Exequiel, Introducción A La Supercomputación En El Peru, vol. 39, no. 5. 2017.

[3] A. S. Carranza Sánchez, J. A. Verduzco Ramírez, N. Farías Mendoza, F. Cervantes Zambrano, and F. Rodríguez Haro, “Plataforma de HPC portable de bajo consumo energético para aplicaciones de minería de datos,” RECI Rev. Iberoam. las Ciencias Comput. e Informática, vol. 6, no. 11, pp. 16–24, 2017.

[4] J. Fiestas, “Construcción e Implementación de un Clúster con máquinas PCs recicladas.,” vol. 14, no. 1, pp. 9–13, 2014.

[5] M. Cruz, “Medidas de rendimiento y comparación entre el Clúster Cruz I y el Clúster Cruz II,” Rev. la Fac. Ciencias la UNI, vol. 17, no. 1, pp. 9–16, 2014.

[6] D. Armando et al., “Computación De Alto Desempeño Para Cálculos De Química Mecano-Cuántica,” p. 3, 2015.

[7] W. Himpe, R. Ginestou, and M. J. C. Tampère, “High Performance Computing applied to Dynamic Traffic Assignment,” Procedia Comput. Sci., vol. 151, no. 2018, p. 411, 2019.

[8] “List Statistics.” [Online]. Available: www.top500.org.

[9] Massachusetts Institute of Technology, “Capítulo 20. Protocolo SSH.” [Online]. Available: https://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html.

[10] M. Brownell, “Building and Improving a Linux Cluster,” 2015.

[11] R. Samir and R. Caro, “Implementación De Un Clúster Experimental Bajo,” p. 12, 2014.

[12] TOP500.org, “Operating System System Share.” [Online]. Available: www.top500.org/statistics/list/.

[13] The Open MPI Project, “A High Performance Message Passing Library,” 2019. [Online]. Available: https://www.open-mpi.org/.

[14] L. Alcántara, “Instalación y configuración de un cluster de alta disponibilidad con reparto de carga,” p. 51, 2014.

[15] D. Jiménez and A. Medina, “Cluster de Alto Rendimiento,” pp. 16–17, 2014.

[16] R. Francisco and A. Moreno, “Escalabilidad de Multiplataforma sobre OpenMPI,” 2016.