Implementation of a Beowulf Cluster and Analysis of its Performance in Applications with Parallel Programming

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*—In the Image Processing Research Laboratory (INTI-Lab) of the Universidad de Ciencias y Humanidades, the permission to use the embedded systems laboratory was obtained. INTI-Lab researchers will use this laboratory to do different research related to the processing of large scale videos, climate predictions, climate change research, physical simulations, among others. This type of projects, demand a high complexity in their processes, carried out in ordinary computers that result in an unfavorable time for the researcher. For this reason, one opted for the implementation of a high-performance cluster architecture that is a set of computers interconnected to a local network. This set of computers tries to give a unique behavior to solve complex problems using parallel computing techniques. The intention is to reduce the time directly proportional to the number of machines, giving a similarity of having a low-cost supercomputer. Different performance tests were performed scaling from 1 to 28 computers to measure time reduction. The results will show if it is feasible to use the architecture in future projects that demand processes of high scientific complexity.**

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

I. Introduction

With the growth of technological advances related to the world of computing, new techniques are emerging that take full advantage of computers that are interconnected by the same local network. The idea is to meet specific needs in less time than an ordinary computer. Performing processes of high availability, efficiency, and performance has become critical to providing better services, optimizing time resulting from complex problems and having continuous availability.

In this work is carried out the implementation of a high-performance cluster type Beowulf, which is the set of low-cost computers interconnected by a network to solve problems of high scientific complexity in less time [1]. This work was done due to the proposal of new projects where the time of the result takes hours or days depending on the complexity of the algorithm. Thanks to this work, one is able to reduce the time in providing results using distributed programming.

The work was done in the embedded systems laboratory of the University of Sciences and Humanities for scientific purposes that would benefit the INTI-Lab to perform simulations of high scientific complexity. These types of architectures are commercially considered supercomputers because they are designed to increase computing power by allowing parallel processing of tasks and high-speed communication [2]. Twenty-eight computers will be used, of which a performance test will be performed using a specialized algorithm. The idea is to measure their scalability and determine the exact number of computers to be used due to bottlenecks that can occur in the communications network.

The computers used in the architecture will continue being of first use for the university student. For that reason, there is a calendar for its scientific use without causing inconveniences at the time of using the laboratory. Reusing the hardware resources for the implementation of this work is not a new idea. One has the Polytechnic University of Altamira [3] that uses the five computers of the optimization laboratory and networks to solve complex problems with a specialized package to measure its scalability. Recycled computers should not be wasted, and a focus can be given to their use in a cluster architecture. As is the case of the National Engineering University [4], which uses recycled computers to obtain a maximum computational benefit. All these architectures that were mentioned are called Beowulf clusters. They are so-called because of the low or regular computing resource [5]. They will use four elements of hardware that are the RAM, the central processing unit (CPU), and its network card.

In this work, the maximum potential of the computers of the laboratory of embedded systems will be used, being the completion of this work a supercomputer with low computer resources.

II. Methodology

The cluster architecture will be implemented in Laboratory 302-B that will use the 29 computers interconnected to a switch. A computer will be in charge of distributing the problem to the 28 remaining computers that will solve a problem applying techniques of computational parallelism. The characteristics of hardware are of low cost; for that reason, the name of cluster Beowulf is chosen that will be detailed with precision in another section of this work. As can be seen in Fig. 1, in the lower-left is specified in the operating system, and the tool one is using for the realization of parallelism between the nodes because they are computers also used for the academic field all will be connected directly to the public network. Concerning the issue of security from the orchestrator computer will generate a unique security key thanks to the Secure Shell protocol, which will be copied to each laboratory computer to prevent an external agent cannot access not having access permissions.

A. Cluster Arquitecture

The cluster architecture can be of three types: high performance cluster (HPC), high availability cluster (HA) and high efficiency cluster (HP) [6] in this work an HPC architecture was implemented that uses powerful tools and processes of computing to generate data in advanced academic research [7], this type of architecture was chosen to take maximum advantage of computing resources in order to obtain successful results in less time, the more project proposals there are the need to use other types of architecture making the future laboratory of the institution a hybrid cluster, the characteristics of the HPC of the laboratory are as follows.

1) Master node: It is the computer in charge of distributing the problem applying parallel programming to distribute it to the slave nodes in order to give a result in less time, in it, one can see the ecosystem of the architecture installing some monitoring packages.

2) Slave node: These computers have two functions, one of them is to take a portion of the general problem that is distributed by the master node and return the final result when it finishes processing, they do not need to have a graphical user interface because it basically needs to be connected to the network to extract the number of cores for the process.

3) Communication network: It is the means that will help the communication between the slave nodes and the master. The better the communication by the network equipment, the lower the network traffic, making the performance of the processes more favorable.

4) Secure shell protocol: In this work, one uses the Secure Shell (SSH) thanks to this protocol the master node can interact securely with the slave node, for information security reasons a unique key was generated from the master node and copies were made to the slave nodes to have a secure cluster architecture.

5) Paralleling tools: To make computers work in parallel requires specialized tools, in this case, was used Open Mpi which is an implementation of open-source message step interface[1], this tool will help the realization of computational parallelism techniques, in the taxonomy of Flynn extracted from [8] shows two types of parallelism.

a) MISD: Applies the technique of multiple instructions to data where its functional units perform different operations on the same data.

b) MIMD: Applies multiple instruction techniques, multiple data used to achieve parallelism, the machines that use these techniques have several processors that operate asynchronously and independently.

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

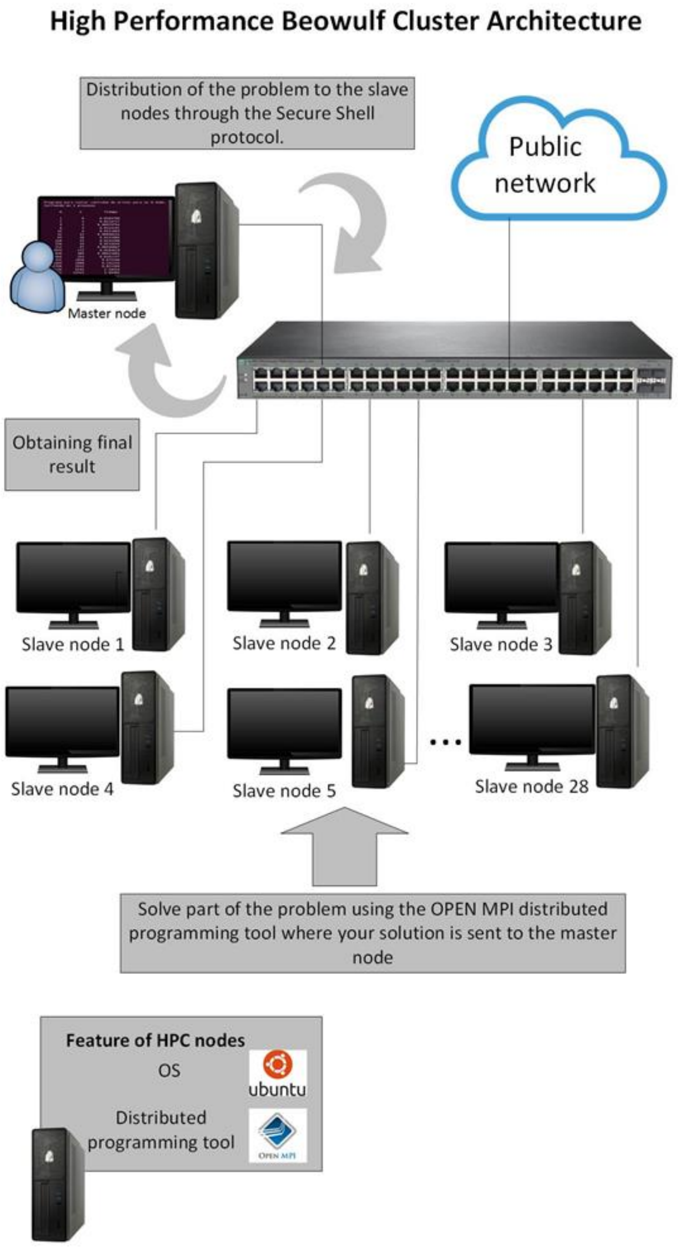
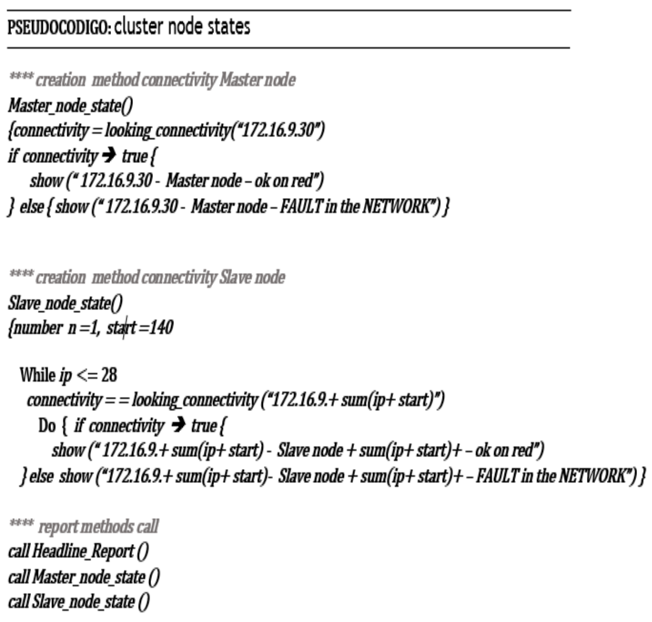

High Performance Beowulf Cluster Architecture
Distribution of the problem to the slave nodes through the Secure Shell protocol.
Public network
Master node
Obtaining final result
Slave node 1
Slave node 2
Slave node 3
Slave node 4
Slave node 5
Slave node 28
Solve part of the problem using the OPEN MPI distributed programming tool where your solution is sent to the master node
Feature of HPC nodes
OS
ubuntu
Distributed programming tool
Open MPI

Fig. 1. Design of the Beowulf Cluster Architecture.

B. Architectural Status Monitoring

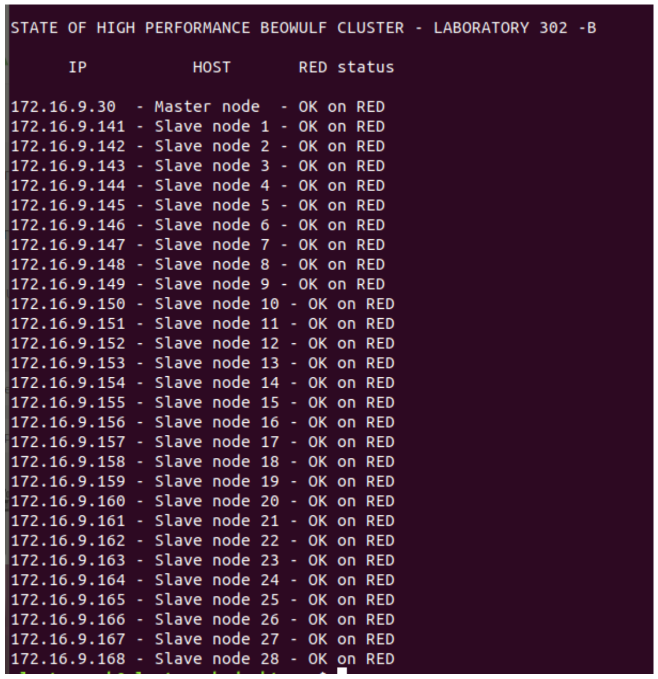
One of the most common cases in the use of a Beowulf cluster is the disconnection that can occur in some of the computers, to access to each one of it to make sure that it has communication can be a tedious work even more in the case of 29 computers, for this problem an algorithm was developed in the programming language Python that shows a report of the nodes with their state of connectivity in Fig. 2 the pseudo code is shown.

As a result, we have the following report, as shown in Fig. 3 that shows 28 slave nodes and one master node, all activated in a network.



```
PSEUDOCODIGO: cluster node states

**** creation  method connectivity Master node
Master_node_state()
{connectivity = looking_connectivity("172.16.9.30")
if connectivity → true {
   show (" 172.16.9.30 -  Master node – ok on red")
} else { show (" 172.16.9.30 -  Master node – FAULT in the NETWORK") }


**** creation  method connectivity Slave node
Slave_node_state()
{number  n =1,  start =140

  While ip <= 28
   connectivity = = looking_connectivity ("172.16.9.+ sum(ip+ start)")
      Do {  if  connectivity → true {
         show (" 172.16.9.+ sum(ip+ start) -  Slave node + sum(ip+ start)+ – ok on red")
  } else  show ("172.16.9.+ sum(ip+ start)-  Slave node + sum(ip+ start)+ – FAULT in the NETWORK") }

**** report methods call
call Headline_Report ()
call Master_node_state ()
call Slave_node_state ()
```

Fig. 2. Pseudo Code Node State of the Beowulf Cluster.



```
STATE OF HIGH PERFORMANCE BEOWULF CLUSTER - LABORATORY 302 -B

      IP           HOST       RED status

172.16.9.30   - Master node   - OK on RED
172.16.9.141 - Slave node 1 - OK on RED
172.16.9.142 - Slave node 2 - OK on RED
172.16.9.143 - Slave node 3 - OK on RED
172.16.9.144 - Slave node 4 - OK on RED
172.16.9.145 - Slave node 5 - OK on RED
172.16.9.146 - Slave node 6 - OK on RED
172.16.9.147 - Slave node 7 - OK on RED
172.16.9.148 - Slave node 8 - OK on RED
172.16.9.149 - Slave node 9 - OK on RED
172.16.9.150 - Slave node 10 - OK on RED
172.16.9.151 - Slave node 11 - OK on RED
172.16.9.152 - Slave node 12 - OK on RED
172.16.9.153 - Slave node 13 - OK on RED
172.16.9.154 - Slave node 14 - OK on RED
172.16.9.155 - Slave node 15 - OK on RED
172.16.9.156 - Slave node 16 - OK on RED
172.16.9.157 - Slave node 17 - OK on RED
172.16.9.158 - Slave node 18 - OK on RED
172.16.9.159 - Slave node 19 - OK on RED
172.16.9.160 - Slave node 20 - OK on RED
172.16.9.161 - Slave node 21 - OK on RED
172.16.9.162 - Slave node 22 - OK on RED
172.16.9.163 - Slave node 23 - OK on RED
172.16.9.164 - Slave node 24 - OK on RED
172.16.9.165 - Slave node 25 - OK on RED
172.16.9.166 - Slave node 26 - OK on RED
172.16.9.167 - Slave node 27 - OK on RED
172.16.9.168 - Slave node 28 - OK on RED
```

Fig. 3. Beowulf Cluster Status Result.

C. Cluster Beowulf

The implementation of the work is of Beowulf type that is denominated with this prefix for the use of components of hardware of low cost that behave as if they were an only computer [9] the computers of the laboratory of embedded systems are used, space where the students of the university carry out their academic activities, therefore, the laboratory has a particular schedule for the accomplishment of investigations, in this architecture was used a number of 28 slave computers and a master computer using a number of 196 cores that will process the problem applying parallel techniques computational parallelism distributing the problem to each of its cores for obtaining results in less time, Fig. 4 shows the computers used in the performance testing process.

All equipment has the same hardware characteristics, as shown in Table I.

D. Performance Testing

In this work, performance tests were performed to measure scalability with an intensive calculation algorithm for the sum of prime numbers used in the C++ programming language.

1) Parallel algorithm for calculating prime numbers: For the performance tests, an algorithm used in previous work with virtual machines was selected [10] that applies parallelism techniques thanks to Open Mpi that performs intensive iterations to each of the numbers to validate if it is a prime number. In this work, an intensive calculation is carried out by testing 2, 4, and 8 million iterations, Fig. 5 shows the pseudocode.

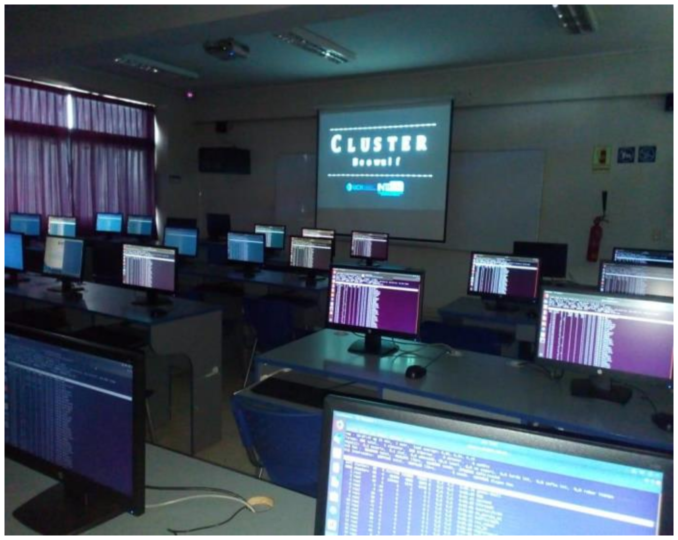
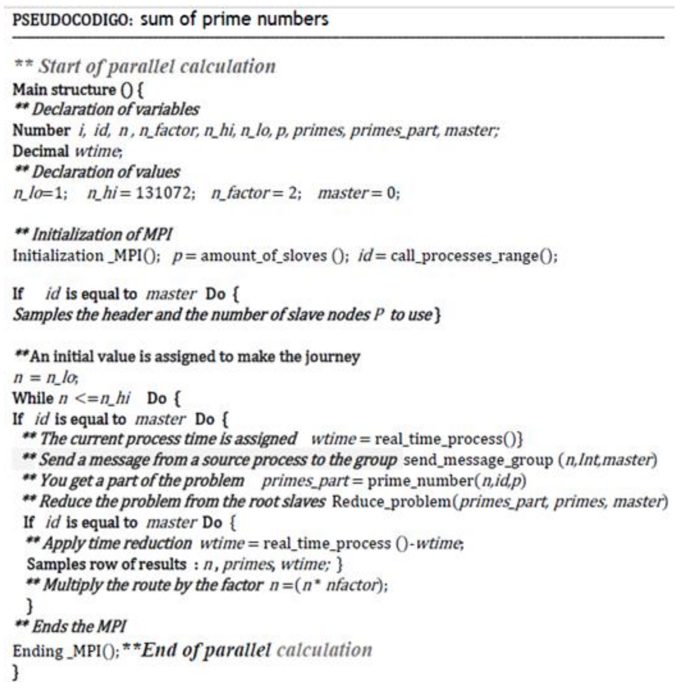


Fig. 4. Comparison of the Performance Test.

TABLE. I. HARDWARE CHARACTERISTICS OF THE BEOWULF CLUSTER COMPUTERS

	Description
Modell	HP EliteDesk 800 G1 SFF
HDD	1 TB
RAM	8 GB
Processor	Intel® CoreTM i7-4790 CPU
Total Cores	7
Type of Operating System	64-bit
Operative System	Ubuntu 18.04



```
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. 5. Pseudo Code of the Cousin Calculation Algorithm.

III. Result

In this section, performance tests are shown using the algorithm of the calculation of prime numbers using distributed programming. Scalability measurements are made using from one to 28 slave nodes of the Beowulf cluster.

For the execution of the algorithm, we will use a line of code from the console of the master node that is: mpirun –np # -hostfile ../.mpi_hostfile ./primos, where the symbol # represents the number of cores and .mpi_host file the number of slave nodes and ./primos the execution of the compiled algorithm. The more slave nodes used, the more kernels must be included from the Linux console to deliver the results in less time.

In Table II, the first tests are performed with 2 million iterations using from one to 28 slave nodes. In Fig. 6, the same result is shown in the form of a statistical graph, taking into consideration the number of slave nodes versus the time of the result.

In Table III, the tests are performed with 4 million iterations using from one to 28 slave nodes. In Fig. 7, the same result is shown in the form of a statistical graph, taking into consideration the number of slave nodes versus the time of the result.

In Table IV, the tests are performed with 8 million iterations using from one to 28 slave nodes. In Fig. 8, the same result is shown in the form of a statistical graph, taking into consideration the number of slave nodes versus the time of the result.

Finally, Fig. 9 shows a statistical graph of the number of slave nodes versus the result time, taking as reference the three performance tests used in the previous tables.

TABLE. II. Intensive Calculation with 2 Million Iterations

Number of slave nodes	Time of result	Core
Slave node 1	92.0369	7
Slave node 2	97.8105	14
Slave node 3	46.722	21
Slave node 4	46.1112	28
Slave node 5	25.8577	35
Slave node 6	46.1088	42
Slave node 7	13.8339	49
Slave node 8	23.1545	56
Slave node 9	15.4168	63
Slave node 10	23.2592	70
Slave node 11	9.33184	77
Slave node 12	23.1032	84
Slave node 13	8.07527	91
Slave node 14	13.2397	98
Slave node 15	11.6152	105
Slave node 16	12.1778	112
Slave node 17	6.44756	119
Slave node 18	15.3812	126
Slave node 19	5.85358	133
Slave node 20	11.5912	140
Slave node 21	10.04327	147
Slave node 22	11.2397	154
Slave node 23	8.14788	161
Slave node 24	4.75896	168
Slave node 25	6.44756	175
Slave node 26	4.48963	182
Slave node 27	3.85358	189
Slave node 28	2.5912	196

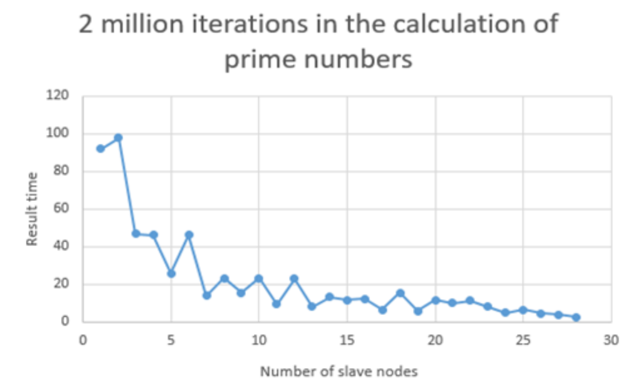


Fig. 6. Statistical Diagram of the Calculation of 2 Million Iterations.

TABLE. III. INTENSIVE CALCULATION WITH 4 MILLION ITERATIONS

Number of slave nodes	Time of result	Core
Slave node 1	351.2	7
Slave node 2	351.09	14
Slave node 3	180.775	21
Slave node 4	178.562	28
Slave node 5	91.6705	35
Slave node 6	176.087	42
Slave node 7	53.1501	49
Slave node 8	88.187	56
Slave node 9	58.6974	63
Slave node 10	88.4901	70
Slave node 11	35.5622	77
Slave node 12	88.2121	84
Slave node 13	31.1266	91
Slave node 14	50.4507	98
Slave node 15	44.3467	105
Slave node 16	44.1493	112
Slave node 17	24.7582	119
Slave node 18	61.3337	126
Slave node 19	22.0287	133
Slave node 20	44.1297	140
Slave node 21	31.1266	147
Slave node 22	22.4177	154
Slave node 23	25.3467	161
Slave node 24	27.4675	168
Slave node 25	19.7582	175
Slave node 26	15.3337	182
Slave node 27	17.6287	189
Slave node 28	13.56297	196

TABLE. IV. INTENSIVE CALCULATION WITH 8 MILLION ITERATIONS

Number of slave nodes	Time of result	Core
Slave node 1	351.2	7
Slave node 2	351.09	14
Slave node 3	180.775	21
Slave node 4	178.562	28
Slave node 5	91.6705	35
Slave node 6	176.087	42
Slave node 7	53.1501	49
Slave node 8	88.187	56
Slave node 9	58.6974	63
Slave node 10	88.4901	70
Slave node 11	35.5622	77
Slave node 12	88.2121	84
Slave node 13	31.1266	91
Slave node 14	50.4507	98
Slave node 15	44.3467	105
Slave node 16	44.1493	112
Slave node 17	24.7582	119
Slave node 18	61.3337	126
Slave node 19	22.0287	133
Slave node 20	44.1297	140
Slave node 21	31.1266	147
Slave node 22	22.4177	154
Slave node 23	25.3467	161
Slave node 24	27.4675	168
Slave node 25	19.7582	175
Slave node 26	15.3337	182
Slave node 27	17.6287	189
Slave node 28	13.56297	196

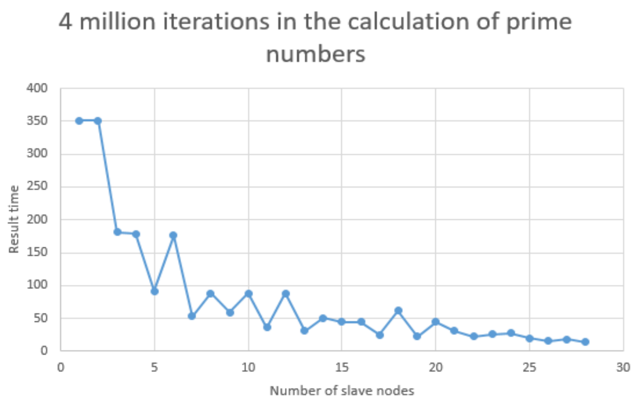


Fig. 7. Statistical Diagram of the Calculation of 4 Million Iterations.

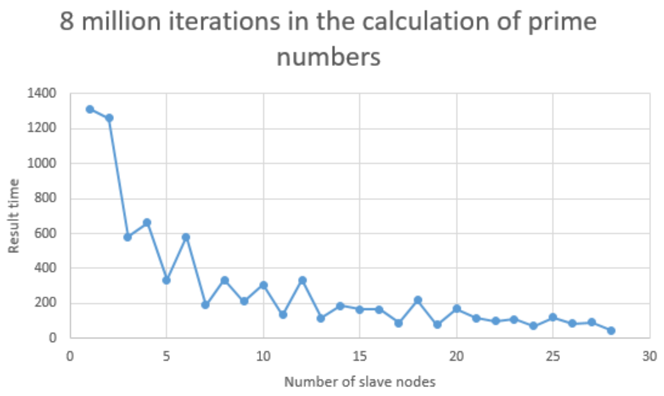


Fig. 8. Statistical Diagram of the Calculation of 8 Million Iterations.

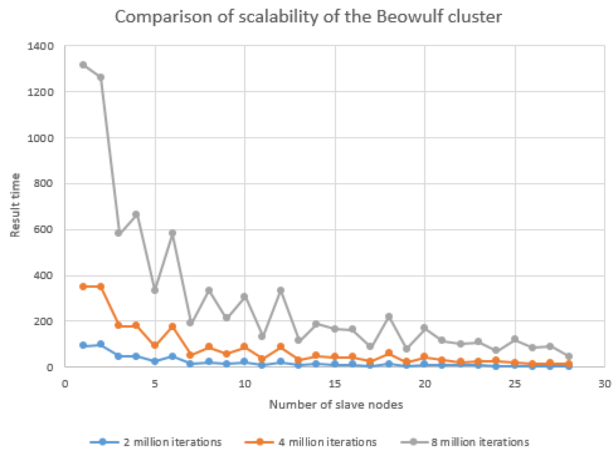


Fig. 9. Comparison of the Performance Test.

IV. Discussion and Conclusions

The work serves as a starting point for the realization of algorithms of high scientific complexity. It has scheduled a schedule of continuous improvement where the activities will be carried out depending on the need that arises in the direction of research of the Universidad de Ciencias y Humanidades. Improvements will include high availability to obtain large volumes of information using Big Data techniques as it does in [11]. This work has similarity concerning the measurement of scalability to use more nodes that demonstrate the efficiencies of these HPC architectures with Big Data Hadoop. Concerning the results section, one sees a reduction in time when more odd nodes are used. This situation is due to its cores that carry out the work in parallel. This work can be improved using a higher number of cores without having to resort to using a new slave node, as one has the case of [12]. This work of the Universidad Nacional de Ingeniería that takes full advantage of the GPU that each computer has demonstrated that the performance is five times higher compared to using CPU.

In future works, related to the increase of the Beowulf cluster potential, it will be proposed to include graphics cards. These graphic cards will make this architecture more powerful using PyCuda.

This work demonstrates that the use of a Beowulf cluster architecture using embedded systems laboratory computers reduces time without the need to acquire specialized equipment. As shown in the results section of Fig. 9, the more complex the problem, the more efficient the slave nodes will be, concluding that the implementation of this architecture meets the proposed objectives.

References

[1] Jiménez and A. Medina, “Cluster de Alto Rendimiento,” Fac. Ing.-UMSA Clust., vol. 1, no. 1, 2014.

[2] J. A. Fiestas-Iquira, “El papel de la supercomputación en la investigación: astrofísica de núcleos galácticos y agujeros negros,” Interfases, vol. 0, no. 008, p. 49, 2015.

[3] M. Vargas-Martínez, S. Gómez-carpizo, J. Sandoval-Sánchez, and G. Castillo-Valdez, “Revista de Sistemas Computacionales y TIC’ s Construcción de clusters de computadoras de bajo costo utilizando software libre Revista de Sistemas Computacionales y TIC ’ s,” Rev. Sist. Comput. y TIC’s, vol. 2, no. 4, pp. 19–25, 2016.

[4] J. Fiestas and C. M. Cruz, “Construcción e Implementación de un Clúster con máquinas PCs recicladas.,” Rev. la Fac. Ciencias la UNI, vol. 14, no. 1, 2014.

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

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

[7] L. Chuquiguanca, E. Malla, F. Ajila, and R. Guamán-quinché, “Arquitectura clúster de alto rendimiento utilizando herramientas de software libre,” vol. 2, no. 1, pp. 1–8, 2015.

[8] A. † Velarde-Martinez, Luna-Ramirez, E. & Haro-Hernandez, and José, “Liebres Inteligentes: Sistema de Multicomputadoras para el procesamiento paralelo de aplicaciones científicas,” Rev. Tecnol. e Innovación, vol. 2, no. 3, pp. 454–463, 2015.

[9] P. Burhanuddin, N. Universitas, M. Indonesia, P. R. View, and P. B. Nuhung, “Cluster Computing Analysis Based on Beowulf Architecture,” Int. J. Comput. Informatics, vol. 1, no. April, pp. 9–15, 2016.

[10] E. L. Huamaní, P. Condori, and A. Roman-gonzalez, “Virtualizing a Cluster to Optimize the Problems of High Scientific Complexity within an Organization,” vol. 10, no. 6, pp. 618–622, 2019.

[11] D. Schmidt, W. C. Chen, M. A. Matheson, and G. Ostrouchov, “Programming with BIG Data in R: Scaling Analytics from One to Thousands of Nodes,” Big Data Res., vol. 8, pp. 1–11, 2017.

[12] N. M. Lapa Romero, J. A. Fiestas Iquira, A. Tenorio Trigoso, and Y. Nuñez Medrano, “Pruebas de rendimiento sobre el Clúster de CPUs y GPUs empleando simulación N-body,” no. July, pp. 19–21, 2018.