Implementation of a Beowulf Cluster in a grid computing environment for teaching parallel programming

Eric Santos de Jesus[1], Isabelle Saahirah Ribeiro de Lima[1], Ariel Andrade Cruz[1], Mikael dos Santos Gois[1], Kauan Gutierrez Santos Estácio[1], Talyson Rodrigues Silva Nascimento[1], José Aprígio Carneiro Neto[2]

[1]Instituto Federal de Sergipe (IFS) – Campus Itabaiana – Sergipe – Brasil

[2]Instituto Federal de Sergipe (IFS) – Campus Nª Senhora do Socorro – Sergipe – Brasil

{ericjesus403}@gmail.com, isabellesaahirah28@gmail.com, arielccomp@gmail.com, mikaelgois1@gmail.com, kauangt2017@gmail.com, talyson.rodigues@gmail.com, jose.neto@ifs.edu.br

Abstract. *This research aimed to implement a Beowulf cluster in a Grid computing environment on the Internet, using Apache Hadoop as a parallelism library and the Docker platform, seeking to spark interest in the scientific community regarding the use of parallel programming architectures and languages. The methodology employed a quantitative, descriptive, and exploratory approach, focused on the practical deepening of the research topic. The results identified that, despite Apache Hadoop being a robust and fault-tolerant library, it still presents gaps that need to be explored and improved, such as the issue of efficient task distribution among the system's nodes. The cluster was fundamental to the students' learning.*

1. Introduction

Over the years, technological advancements have driven the need for increased computer processing power, contributing to the emergence of different types of computer architectures that are more complex and robust, capable of performing highly complex tasks with large volumes of data.

However, acquiring machines with these capabilities and architecture requires high investments. Therefore, a viable alternative to minimize this problem is the implementation of high-performance, low-cost clusters, formed by non-specialized, homogeneous or heterogeneous personal computers (Beowulf).

To maximize the processing efficiency of this type of cluster, it is essential to use parallel programming libraries that can operate efficiently, simultaneously, and in a distributed manner to execute complex tasks involving large volumes of data. Furthermore, these libraries need to be fault-tolerant and offer an abstraction for the creation and management of threads and processes, such as the Apache Hadoop library.

However, using these libraries presents some challenges, including: the use of different types of communication and synchronization primitives; distinct process or thread abstractions; different programming models; load balancing problems; and the impact on the development and performance of the developed applications, depending on the type of platform and parallel library used to solve the problems to be addressed.

Thus, in an attempt to overcome the challenges and demands at hand, this article aimed to propose the implementation of a Beowulf cluster within a grid computing environment, utilizing Apache Hadoop as a parallelism library and the Docker platform, with the goal of sparking the scientific community's interest in the use of parallel architectures and programming languages.

2. Theoretical Framework

2.1. Parallel Architectures

Parallel architectures emerged as a solution to meet the growing demand for data processing and the execution of complex tasks. In contrast to traditional systems, which execute one task at a time, these architectures allow multiple operations to be performed simultaneously by multiple processors or processing cores, increasing computational performance, reducing application execution time, and enabling the development of advanced solutions in various fields (Yi, 2024).

However, to achieve parallelism, the processes or control flows must be executed simultaneously and in a distributed manner, making the use of computers with multiple cores or a set of them, called a cluster, essential (Navaux et al., 2011).

In this context, the evolution of parallel architectures has enabled the creation of various types of computing environments that are increasingly robust, scalable, and flexible, optimizing task processing time, especially for tasks involving high-performance computing.

2.2. Cluster Beowulf

Beowulf clusters are formed by personal computers, organized in a parallel and distributed computing environment, where each processing node functions as a parallel processor, executing parts of a larger task, working together to perform complex, high-performance calculations, such as: scientific simulations, modeling, image rendering, analysis of large volumes of data, among others (Datti et al., 2015).

The performance of a Beowulf cluster depends on the communication capabilities between the machines on the network and on efficient load balancing between them, since frequent data exchange between processes can become a limiting factor, making the use of high-speed networks and scheduling mechanisms that can distribute tasks more evenly among the equipment essential (Rauber & Rünger, 2013).

In a Beowulf cluster, composed of heterogeneous machines, there is a variation in hardware resources between the nodes, such as: differences in processing capacity, memory, and storage capacity. These factors make load balancing and dynamic task allocation a challenging scenario, but they allow for the adaptation of processing to the different capacities of the nodes that make up the cluster (Pereira, 2016; Maia, 2016).

The structure of a Beowulf cluster is composed of a series of essential components, such as: a cluster node (an independent computer); a high-speed network (essential for data transfer between the cluster nodes); an operating system (robust, scalable, and efficient); and a communication library that enables task parallelism and efficient communication between nodes, as well as task distribution and distributed data storage, and is fault-tolerant (Datti et al., 2015).

According to Datti et al. (2015), the basic operation of a Beowulf cluster is divided into three stages: division of labor, where the large and complex problem is divided into small subproblems that can be executed independently on different nodes of the cluster; parallel execution, where each node receives a subproblem and executes it in parallel; and finally, communication and aggregation, which allows the interconnection between the nodes of the cluster through the network, enabling the efficient exchange of information between them while the processing of the subproblems is being carried out.

2.3. Apache Hadoop Library

Apache Hadoop is an open-source parallelism library used to execute parallel tasks that require high computer processing power for handling large volumes of data. This library is responsible for managing the resources of the computers that make up the cluster, as well as coordinating the execution of data processing algorithms (Menezes et al., 2016).

The Apache Hadoop framework has essential elements, such as: HDFS (Hadoop Distributed File System), the system responsible for file storage; and MapReduce, the framework responsible for processing data, which divides the work into small parts called "map" and "reduce". In addition, there are some other components that assist during the execution and processing of parallel algorithms (Goldman et al., 2012).

Apache Hadoop is software that provides reliable data storage, possessing a simplified management structure, as the tool itself is responsible for managing communication between cluster nodes and distributing tasks during data processing. It also offers scalability, as adding new nodes to the cluster doesn't require significant configuration changes (Menezes et al., 2016).

However, some problems can be encountered when using it, such as: difficulty in finding faults, since several log files are created during task execution, scattered across different system folders; the processing of small files, as the library is not well adapted to this type of situation; and the existence of only one master node, which creates a critical point for the cluster's operation (Goldman et al., 2012).

Combining Apache Hadoop with heterogeneous Beowulf clusters increases the scalability and efficiency of parallel processing, while also allowing for a more balanced use of computing resources, respecting the capacity differences between nodes (White, 2015).

2.4. Docker

Docker is a container platform that enables the creation, distribution, and execution of applications in an isolated manner, allowing developers to easily create and distribute containers via command line (Docker CLI) or through configuration files, called Dockerfiles (Merkel, 2014).

A container consists of a lightweight and portable unit that contains everything necessary for the execution of an application, such as dependencies, libraries, and the application code itself, allowing the application to be run identically in different environments, eliminating problems related to lack of version compatibility and environment configurations (Merkel, 2014).

One of the advantages of Docker is the cache generated during image creation. An image is a read-only model used to characterize the container; it can be customizable or based on an existing one. Thus, it's possible to create a container based on an image of the Linux operating system, Ubuntu distribution, among others, but with the help of other applications, such as an Apache server. The image is created from the Dockerfile, a file that contains instructions necessary for the container's characterization. Each instruction in a Dockerfile becomes a layer, and when these layers are changed, only those layers are rebuilt. Therefore, caching is used to reduce build time, avoiding the reconstruction of the entire image (Dockerdocs, 2025).

Data persistence in Docker is achieved through volumes, which are directories mounted on the host where the container is running, but accessible from within the container. Data saved in the container is also saved on the host. Therefore, if one container fails, another will be started pointing to the same volume, and the data from the previous container will be available to the new one (Dockerdocs, 2025).

When applied to a Beowulf cluster, Docker offers an easier way to manage runtime environments for different applications and services. Containers reduce system overhead and facilitate dynamic scalability, as well as allowing for more efficient execution on machines with different configurations (Merkel, 2014).

According to Merkel (2014), Docker and containers represent a revolution in the way applications are developed, distributed, and executed, bringing a series of benefits such as portability, load balancing, efficiency, and consistency, making life easier for developers.

In this project, Docker and containers worked together to try to reduce the load balancing problems between the cluster nodes, caused by Apache Hadoop, by making more efficient use of all the hardware resources available in the system.

3. Methodology

The methodology used in this article had a quantitative, descriptive, and exploratory approach, aimed at the practical deepening of the research topic, making it clearer and leading to the construction of new knowledge with the implementation of a cluster.

Initially, a review of the state of the art was conducted, providing an overview of the research topic, with the aim of understanding the main concepts related to parallel computing and parallelism libraries. Subsequently, a Beowulf cluster was implemented, consisting of four heterogeneous machines with the following physical characteristics: Master (LENOVO IdeaPad 6 notebook, AMD Ryzen 7 5700U CPU @ 4.37 GHz, AMD Lucienne GPU and 6 GB of RAM); Slave 1 (Desktop, AMD Ryzen 3 3100 CPU @ 3.90 GHz, NVIDIA GeForce RTX 2060 Rev. A GPU and 16 GB RAM), both located in the city of Lagarto – Sergipe – Brazil, at different addresses, connected to the internet via a 100 MB link to two other slave machines, located in the city of Itabaiana – Sergipe – Brazil, at different addresses, connected to the internet via a 400 MB link, with the following physical characteristics: Slave 2 (Desktop, AMD Ryzen 5 3400G CPU @ 3.70 GHz, AMD ATI Radeon RX 580 2048SP GPU and 24 GB RAM); Finally, Slave 3 (Desktop, Intel i3-2330M CPU (4 threads) @ 2.20GHz, Intel 2nd Generation Core Processor Family GPU, and 8 GB of RAM). All machines were interconnected to the cluster via a Virtual Private Network (VPN), whose architecture of the implemented Beowulf Cluster can be seen in Figure 1.

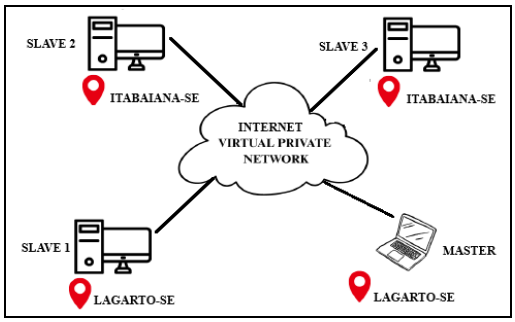


Figure 1. Beowulf Cluster Architecture

The following software tools were used to configure the cluster: Apache Hadoop (3.4.0), Java (OpenJDK) 8 (openjdk-8-jdk), Docker (28.1.1, build 4eba377327), ZeroTier (1.14.2), Zabbix (7.0 LTS (server, agent2, frontend)), MySQL Server (Ubuntu 22.04 default version (8.0)) and Grafana (Enterprise 11.2.0).

The cluster performance tests involved running the following parallel computational algorithms: TeraGen, TeraSort, and TeraValidate, with a file size of 50 GByte. To evaluate cluster performance, the following metrics were used: CPU, RAM and Network Traffic. The metrics were collected using Zabbix and Grafana.

4. Results and Discussion

This section presents the process of creating and configuring the cluster, as well as the results of the benchmark tests performed and the cluster performance metrics collected through Zabbix and displayed graphically via dashboards (Grafana).

4.1. Cluster Creation and Configuration

The following will show the process of creating and configuring the cluster implemented in this work:

1. The process begins with the creation of an account in the ZeroTier tool, a basic account with configuration rights for up to 5 machines on the network, free of charge, through the tool's official website (https://www.zerotier.com).
2. Creation of a network in ZeroTier;
3. Definition of the IPv4 address block to be used in the network, with the identification of the IP of each host to the cluster;
4. Installation of ZeroTier on the Linux Ubuntu distribution;
5. Joining the network created by ZeroTier, through its ID;
6. Installation of the Docker tool;
7. Creation of the cluster and testing of the machines configured in Docker;
8. Running the container with the native network (host mode);
9. Starting the services of the tools: OpenBSD SSH Server; MySQL Server; Grafana; and Apache Hadoop;
10. Creation of the Apache Hadoop cluster automation script;
11. Execution of the Apache Hadoop cluster automation script;
12. Centralized initialization of the Apache Hadoop services: NameNode, DataNode, ResourceManager, and NodeManager.

After initializing the services, it is necessary to verify if the Apache Hadoop cluster ports are accessible (Table 1).

Table 1. Beowulf Cluster Ports and Services

Ports	Services	URLs	Description
9870	HDFS NameNode UI	http://master:9870	Interface for viewing the HDFS distributed file system
9864	HDFS DataNode UI	http://slave1:9864	Detailed interface of the local storage for each DataNode
8088	YARN If any problem occurs UI	http://master:8088	Interface for managing jobs and resources in the cluster

4.2. Benchmark Tests

In this stage, the results of benchmark tests performed on the cluster will be presented, involving the parallel computational algorithms: TeraGen (data generation), TeraSort (data organization), and TeraValidate (data organization validation), in addition to performance metrics. A 50 GByte file was used for the test scenarios.

During the test, the following cluster performance metrics were collected using Zabbix software: CPU Utilization, Memory Utilization, Bits Sent, Bits Received, Available Memory.

4.3. TeraGen Algorithm

TeraGen is responsible for generating random input data for processing in the cluster, through the execution of a MapReduce (Job MapReduce), which generates random binary data with keys and values of different sizes. This algorithm uses 100-byte lines as input parameters for data generation. Thus, the algorithm divides the desired number of lines by the number of tasks, assigning line intervals to each Mapper, and this, in turn, modifies the random number generator to the correct value of the first line and then generates the following lines (Alves, 2015).

The results of the cluster performance test, as well as the behavior of each machine in the system, can be seen in Figure 2.



Figura 2. Cluster performance metrics (TeraGen)

The results showed low CPU usage by the cluster nodes, but higher usage by slave node 3. This occurred because, despite the use of Docker and containers, the

Apache Hadoop library randomly selected a cluster node (slave) to perform an initial task processing function, regardless of the machine's physical capacity, consuming more hardware resources from that machine and causing an imbalance in the system.

Regarding RAM usage, a scenario similar to the previous metric was observed; that is, despite consumption reaching approximately 50% on one of the system nodes, RAM resources remained unbalanced, leaving some machines in the cluster underutilized. This highlights the inefficiency of Apache Hadoop in balancing and distributing the system load, despite the use of containers.

In network traffic, a significant and balanced volume of information exchange was observed between the cluster nodes during task execution, both in terms of bits sent and received, reaching a maximum volume of approximately 40 Mbytes in both situations. The analysis of this metric revealed a different behavior for the library, demonstrating greater efficiency in load balancing and task distribution within the system with respect to this metric.

4.4. TeraSort Algorithm

TeraSort is an algorithm that uses a sample of data randomly generated by TeraGen and, through a Map/Reduce program, sorts and orders this sample. To perform this task, TeraSort creates a sorted list of *N-1* keys, where *N* is the number of Task Reduces that defines the key range for each Task Reduce. Thus, the keys generated by TeraGen follow a list *[i-1]<=key<list[i]*, which are sent to the execution of Reduce *i*, sorting the keys received in each Reduce. The outputs of each Reduce *i* are then assembled following the order of *i* (O'Malley, 2008).

The TeraSort algorithm is used to evaluate large-scale sorting and ordering performance, test MapReduce's parallelism capabilities, measure cluster throughput, and perform comparative benchmarks between different cluster or hardware configurations.

The results of the cluster performance metrics, related to the TeraSort algorithm tests, can be seen in Figure 3.

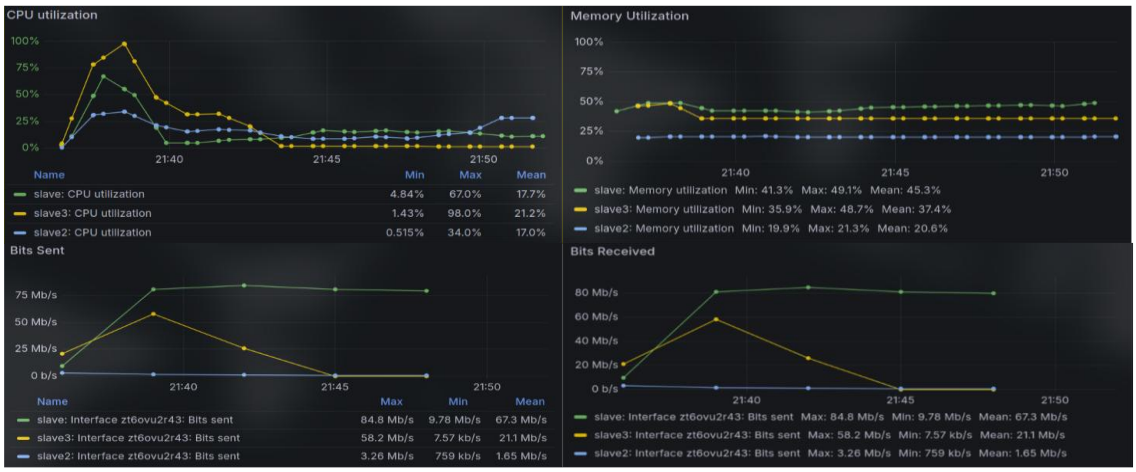


Figura 3. Cluster performance metrics (TeraSort)

The most intensive CPU usage was evident in Slave 3, which reached almost 100% of this resource, considering the complexity of the algorithm and the effort expended by the system to solve the problem. However, underutilization of these

hardware resources was observed in the other nodes that make up the system, due to the aforementioned problem of lack of load balancing by the library.

The RAM usage metric followed the same pattern as CPU consumption, with one of the system nodes (Slave 1) being used more than the others, resulting in the underutilization of these hardware resources by the system.

Since the TeraSort algorithm demanded significant hardware resources from the system, the network traffic scenario was no different. Significant, yet unbalanced, data traffic was observed in both sending and receiving, with two nodes being more heavily used: slave nodes 1 and 3. The maximum sending traffic reached approximately 85 Mb/s, reaching the same value in receiving.

4.5. TeraValidate Algorithm

TeraValidate is a Map/Reduce algorithm that verifies whether the output data, generated by TeraSort, has been sorted correctly. To perform this validation, the algorithm creates a Mapper for each file in the output directory, with each Mapper ensuring that each key is less than or equal to the previous key. Additionally, the Mapper creates records with the first and last keys of the file, where the Reduce function ensures that the first key of file *i* is greater than the last key of file *i - 1*. If any problem occurs during the validation of the Reduce outputs due to incorrect key ordering, the program will report an error (Neto et al., 2022).

Considering its algorithmic nature, which was validation, the TeraValidate algorithm consumed few system hardware resources. During its execution, the maximum CPU resource utilization in the cluster, on one of the nodes (Slave 1), reached approximately 30%, while on the others, it reached approximately 10% (Slave 3) and 5.7% (Slave 2), respectively. Therefore, a scenario of imbalance in the load distribution among the system nodes was again observed, with one node being more utilized than the others during task processing (Figure 4).



Figura 4. Cluster performance metrics (TeraValidate)

Regarding RAM usage, an unbalanced consumption scenario was identified, with slave machines 1 and 3 being more heavily used during task execution, as well as

underutilization of hardware resources in slave 2, with a maximum consumption of 17%. Despite the algorithm being merely a validation algorithm, without much complexity, it demanded practically half of the RAM resources of the nodes that actively participated in task processing (Figure 4).

During the execution of the TeraValidate algorithm, despite its low hardware resource requirements, a highly disproportionate network traffic behavior was observed between the nodes. While slave 2 consumed approximately 100Mb/s of available network resources in sending and receiving data, the other nodes in the system (slaves 1 and 3) barely managed to reach 1Mb/s. This further highlights the Apache Hadoop library's preference for one machine in the cluster, causing an imbalance in the system (Figure 4).

5. Conclusion

This work aimed to show that it is possible to reduce barriers in computer architecture education, with regard to the use of parallel architectures and distributed systems, through the implementation of clusters that use common machines (Beowulf) in their construction, whether homogeneous or heterogeneous, located geographically close or distant, with the hardware resources available to students and teachers, thus meeting the need and costs of acquiring supercomputers or more robust machines for this type of academic practice.

Furthermore, this project contributed, both theoretically and practically, to the teaching of parallel and distributed programming, identifying the main methodologies, architectures, and computational tools used in the teaching-learning process of this type of programming, as well as in learning libraries and programming languages used in parallelism, considering that this type of programming allows for making the most of the processing power available in multicore architectures, aiming to solve problems that present greater computational complexity.

The implemented cluster was built with only a few machines, solely for the purpose of demonstrating, in practice, the operation and behavior of a parallel architecture, as well as the performance of parallelism libraries used in this type of implementation. However, these dynamic allowed students a greater understanding of the concepts involved in parallel and distributed computing.

The test results revealed that, despite Apache Hadoop being a robust and fault-tolerant library, it still has gaps that need to be explored and improved, such as the efficient distribution of tasks among the system's nodes and load balancing.

As future work, it is recommended to conduct new test scenarios involving a larger number of machines in the cluster, as well as the use of other computational algorithms, in order to observe the behavior of the Apache Hadoop library as the number of machines in the system scales.

Acknowledgments

We would like to thank the Foundation for Support of Research and Technological Innovation of the State of Sergipe – FAPITEC/SE, the State Secretariat for Economic Development and Science and Technology – SEDETEC/SE, and the Federal Institute of Sergipe – IFS, for their financial support in the development of this research project.

References

Alves, N. D. M. (2015). *Uma abordagem não intrusiva e automática para configuração do Hadoop*. https://www.repositorio.ufal.br/handle/riufal/1719.

Datti, A. A., Umar, H. A., & Galadanci, J. (2015). A Beowulf cluster for teaching and learning. *Procedia Computer Science*, *70*, 62-68. DOI: 10.1016/j.procs.2015.10.034

Dockerdocs. *Visão geral de redes*. (2025). https://docs.docker.com/engine/network/.

Goldman, A., Kon, F., Junior, F. P., Polato, I., & de Fátima Pereira, R. (2012). Apache Hadoop: conceitos teóricos e práticos, evolução e novas possibilidades. *XXXI Jornadas de atualizações em informática*, 88-136.

Maia, R. M. S. (2016). *Ferramenta para criação de clusters virtuais para o ensino de programação paralela e distribuída*. http://repositorio.ufc.br/handle/riufc/24785

Menezes, S. L., Freitas, R. S., & Parpinelli, R. S. (2016). Mineração em grandes massas de dados utilizando hadoop mapreduce e algoritmos bio-inspirados: Uma revisão sistemática. *Revista de Informática Teórica e Aplicada*, *23*(1), 69-101.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux j*, *239*(2), 2.

Navaux, P., D. R., C., & Pilla, L. L. (2011). Fundamentos das arquiteturas para processamento paralelo e distribuído. *XI Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul-2011-Porto Alegre, RS*, 22-59.

Neto, A.J.A, Neto, J.A.C, & Moreno, E.D. (2022). O desenvolvimento de um cluster de big data de baixo custo usando Apache Hadoop e Raspberry Pi. Um guia completo. *Computers and Electrical Engineering*, *104*, 108403.

O'malley, O. (2008). Terabyte sort on apache hadoop. *Yahoo, available online at: http://sortbenchmark. org/Yahoo-Hadoop. pdf,(May)*, 1-3.

Pereira, V. (2016). *Instalação e configuração de um Cluster Beowulf*. Escola Superior de Tecnologia e Gestão. Instituto Politécnico da Guarda. Portugal. http://hdl.handle.net/10314/3697

Rauber, T., & Rünger, G. (2013). Parallel programming: For multicore and cluster systems. [sl]: *Springer science & business media*.

Yi, X. (2024). A study of performance programming of cpu, gpu accelerated computers and simd architecture. *arXiv preprint arXiv:2409.10661*. DOI: 10.48550/arXiv.2409.10661

White, T. (2015). *Hadoop: The definitive guide*. 4. Ed. " O'Reilly Media, Inc.". ISBN: 978-1-449-35853-1. 2015.