Computing and Artificial Intelligence 2025, 3(1), 2018.
https://doi.org/10.59400/cai2018

Article

Intelligent process migration in heterogeneous distributed systems

Terecio Diosnel Marecos Brizuela[1], David Luis La Red Martínez[2,*], Federico Agostini[2], Jorge Tomás Fornerón Martínez[1]

[1] Faculty of Applied Sciences, National University of Pilar, Pilar 120101, Paraguay
[2] Faculty of Exact and Natural Sciences and Surveying, Northeastern National University, Corrientes W3400, Argentine
*** Corresponding author:** David Luis La Red Martínez, lrmdavid@exa.unne.edu.ar

CITATION

Marecos Brizuela TD, La Red Martínez DL, Agostini F, Fornerón Martínez JT. Intelligent process migration in heterogeneous distributed systems. Computing and Artificial Intelligence. 2025; 3(1): 2018. https://doi.org/10.59400/cai2018

ARTICLE INFO

Received: 11 November 2024
Accepted: 9 December 2024
Available online: 18 December 2024

Abstract: In distributed processing environments, multiple groups of processes are found sharing resources and competing for access. These processes may or may not require synchronization and it is essential to reach a consensus to manage access to resources in a way that establishes a strict order, thus ensuring mutual exclusion. The proposal presented is an innovative and effective solution for the management of shared resources in distributed systems, which allows solving problems related to overload and workload balancing. The evaluation of the state of computational loads and the final comparison using standard deviation are useful tools to detect and correct imbalances in the system. In addition, the possibility of establishing initial configurations of the algorithm for each particular situation allows adapting the solution to the specific needs of each system.

Keywords: migration; load balancing; synchronization; distributed systems; mutual exclusion

1. Introduction

In distributed processing systems, the need to coordinate the allocation of shared resources for processes in a mutually exclusive environment is common. This involves the task of determining the order in which shared resources should be allocated to the processes that require them, as described in La Red Martínez [1].

Each node in a given system has a specific capacity of computational, data and communication resources. Resource requesting processes can be classified into different categories, such as computation intensive, data intensive, and communication intensive, according to their resource requirements.

In many cases, nodes may experience overload when trying to handle multiple resource allocations to processes, resulting in slow node response to user requests. On the other hand, some nodes may remain idle for much of the time. For all these reasons, a proper workload balance is sought by performing intelligent process migrations to improve system performance in heterogeneous environments.

Achieving effective load balance involves redistributing the workload among the nodes of the distributed system in order to improve resource utilization and response times. In turn, it prevents some nodes from being heavily loaded and others from being idle or with little work. A load balancing algorithm must have prior knowledge about the behavior of tasks and the global state of the system. Load balancing decisions are based on global information about the current state of the system, and their effective development covers important aspects such as load estimation, comparison of load and performance levels, system stability, information exchange between nodes, estimation of task resource requirements, selection of tasks for transfer, checking compatibility between nodes, choice of remote nodes, among others.

Based on the aforementioned, the objective of this work is to propose a decision model for a heterogeneous distributed system that integrates the intelligent migration of processes to the aggregation operator described in La Red Martínez [1]. This will allow the redistribution of the workload among the nodes of the system, improving resource utilization, processing response times and increasing performance.

In the Related Works section, various articles addressing topics related to this work have been discussed, providing a global perspective on different approaches and aspects connected to the problem outlined in this brief Introduction. However, it is noted that this work will focus on the application of the decision model and the aggregation operators proposed in La Red Martínez [1] for the type of distributed systems described in the Scenario section.

2. Related works

Among the methods considered traditional for resource allocation in distributed systems respecting mutual exclusion in the access to shared resources, the ones mentioned in Ricart and Agrawala [2], Guohong and Singhal [3] and Lodha and Kshemkalyani [4] stand out.

Innovative methods with respect to resource allocation to processes are mentioned in La Red Martínez [1] and La Red Martínez et al. [5,6], and an alternative with respect to resource allocation, considering strict levels of consensus and sorting by process groups can be seen in Agostini and La Red Martínez [7], Agostini et al. [8] and Agostini [9].

There are different studies on process migration methods, load balancing and energy consumption. In Beiruti and Ganjali [10], they develop a new protocol that is used for a wide range of network applications, such as load balancing, energy saving and resource optimization.

Upadhyay and Lakkadwala [11] develop a migration algorithm to migrate overloaded processes from one machine to another in the same cloud environment considering resource utilization.

In Deshmukh and Deshmukh [12] a modified approach is adopted to create a new distributed dynamic load balancer for distributed file system, the algorithm considers constraints such as network load, disk IO (input/output) load, disk capacity and load which was not considered in other approaches.

In Junaidi et al. [13] a machine learning algorithm is proposed that uses server resources, CPU and memory, to predict the future of server loads.

Design methodologies to quantitatively predict migration performance and power consumption are investigated in Liu et al. [14] and analytical models, based on stochastic reward networks (SRNs), are proposed in Asadi et al. [15] to analyze the impact of resource allocation algorithms and process migration methods on power consumption and performance of virtualized systems.

Some decision models using appropriate aggregation operators are shown in Chiclana et al. [16] and Dong et al. [17].

To define the intelligent process migration methods for resource and process management in DS in this paper, a systematic review of the related literature has been performed, which allowed determining which ones were the most appropriate for the

different scenarios being studied, having as reference more relevant research on process migration methods, which are discussed below. In Sohrabi and Mousavi Khaneghah [18], the challenges of occurrence of dynamic and interactive events in virtual machine-based process migrators are examined by analyzing the migrator performance function.

A comprehensive survey of existing load balancing and job migration techniques is presented in Rathore and Chana [19], where a detailed classification based on different parameters depending on the analysis of existing techniques is included, and a new load balancing technique has been proposed and discussed.

In Chang et al. [20], a resource-aware edge process migration (REM) scheme is proposed which can optimize the process migration decision. In Marecos et al. [21], the application of a competency-based and student-centered learning model is proposed for the study of controlled migration of processes, and where the student must analyze the behavior of resources and processes under different workloads, considering load balancing as the main objective. In Fornerón Martínez et al. [22] an aggregation operator for shared resource management in distributed systems using 2-tuples associated with linguistic labels is proposed.

In Bishop et al. [23], the complexity and challenges of migrating processes in heterogeneous distributed systems, where hosts vary in architecture and operating systems, are addressed. The document presents a valuable contribution to the field of distributed systems but faces technical challenges in achieving a fully functional and automated implementation.

In Cao et al. [24], an innovative and promising proposal is presented, but the impact of the model in terms of energy consumption -a key factor in mobile and intelligent devices- is not explored in depth. Additionally, the reliance on deep learning could require additional computational resources, potentially contradicting the efficiency objectives in edge environments.

Refers specifically to the migration of business information systems from a local environment to a cloud computing platform, rather than process migration within a cloud environment [25]. It focuses on using stochastic models to analyze and facilitate this transition, highlighting challenges in aligning business goals with IT system constraints.

Explores the transformative role of artificial intelligence (AI) in cloud computing, emphasizing its impact on scalability, resource management, and predictive analytics within distributed systems [26]. As organizations increasingly adopt cloud infrastructure, the article asserts that AI technologies are becoming indispensable tools for improving performance, efficiency, and system adaptability.

In Devan et al. [27] cloud migration refers to the process of transferring data, applications, and IT infrastructure from on-premise data centers to a cloud environment. This transition unlocks the benefits of cloud computing, including scalability, agility, cost-efficiency, and access to advanced services. This paper does not refer to the migration of data and processes of a distributed system with mutual exclusion in access to resources, in an already operational cloud environment.

In Rathore and Chana [28] addresses a vital issue in grid computing -load balancing and job migration-. While it provides a comprehensive review of existing

challenges and gaps, it lacks specific solutions and detailed analysis to advance the field. The discussion on the need for innovative load balancing algorithms is valid.

Proposes a two-stage genetic mechanism for load balancing virtual machine hosts (VMHs) in cloud computing environments through migration [29]. It integrates two genetic-based methods: a) Performance Modeling with GEP: Gene Expression Programming (GEP) is used to generate symbolic regression models that predict virtual machine (VM) performance based on their parameters. These models estimate the VMH loads after load balancing; b) Optimal VM Assignment with Genetic Algorithms: Using the load estimates from GEP, a genetic algorithm determines the optimal assignment of VMs to VMHs, considering both current and future loads to achieve effective load balancing. The approach was tested in a real cloud computing environment (Jnet) and implemented as a centralized load balancing mechanism. Experimental results demonstrate that the proposed method outperforms traditional methods like heuristics and statistical regression in achieving better load balancing. The proposed methods consider virtual machines migration, but do not take into account the migration of processes and resources considering mutual exclusion of shared resources.

Introduces a hybrid load balancing algorithm for cloud computing, called Clustering-Based Multiple Objective Dynamic Load Balancing (CMODLB), combining supervised (artificial neural networks), unsupervised (clustering), and soft computing (interval type 2 fuzzy logic system) techniques [30]. This hybrid approach ensures optimal scheduling, efficient migration, and improved load balancing for PMs and VMs in cloud environments. This proposal only considers the migration of virtual machines but does not address the migration of processes and resources between virtual machines.

Introduces a game-theoretical framework for achieving user-optimal load balancing in heterogeneous distributed systems [31]. The static load balancing problem is modeled as a non-cooperative game among users, where the Nash equilibrium represents the optimal operational point for each user. A distributed algorithm is derived based on the structure of the Nash equilibrium to compute the load balancing solution. This proposal does not address load balancing under uncertainty or dynamic scenarios, nor does it consider collaborative process groups.

Presents a novel method for solving the load balancing problem in distributed computer systems using game theory and a genetic algorithm [32]. The load balancing problem is modeled as a non-cooperative game among users, where each player's strategy involves determining the fraction of their job assigned to system computers. The goal is to minimize users' expected response time by maximizing job processing. A genetic algorithm, based on Nash equilibrium, is used to find near-optimal strategies. The method effectively improves performance by reducing response times and ensuring fairness. The possibility of collaborative process groups sharing resources in a distributed mutual exclusion mode is not considered.

Analyzes the traffic load distribution in mesh networks and proposes a fault-tolerant Network-on-Chip (NoC) design that focuses on load balancing [33]. The core idea is to allocate a varying number of Error Correction Code (ECC) decoder modules to each router based on traffic load distribution, aiming to improve ECC decoder module utilization and reduce area overhead without compromising fault tolerance.

This interesting proposal only considers traffic, aiming to optimize it, but it does not take into account the processing load of the different nodes in the distributed system.

Proposes an Efficient Load Balancing algorithm (ELB) for NoC mapping, which considers the relationships between tasks and resource utilization [34]. This proposal does not consider the memory load and input/output operation load on the distributed nodes. It also does not explicitly consider mutual exclusion in the use of shared resources.

Explores the application of self-organizing algorithms, originally used in wireless sensor networks, to networks-on-chip (NoCs), and introduces a method for assigning hierarchical coordinates and a greedy forwarding algorithm for pathfinding [35]. It focuses on handling the complexities of NoC environments, including network overload and the presence of fault nodes, and provides a set of rules for bypassing blocked sections of the network. This proposal does not take into account processing load, memory usage, or input/output operations in heterogeneous distributed systems with distributed mutual exclusion requirements for accessing shared resources.

In Gogoi et al. [36] the study addresses faults in multi-core architectures and Network-on-Chip (NoC) systems caused by aggressive communication workloads. It proposes a fault-aware routing approach that selects paths based on the health of neighboring routers. Additionally, a router cool-off mechanism is introduced to manage workload thresholds. A new router micro-architecture supports these strategies, achieving zero fault occurrence, improved throughput, and minimal delay compared to existing methods. The proposal does not consider resource and process management in a distributed system composed of a set of heterogeneous nodes connected by a wide-area network.

3. Scenario

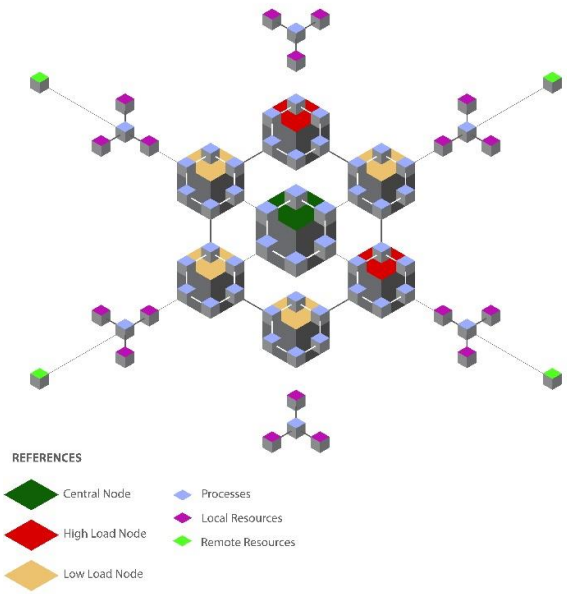

Figure 1. Distributed system.

In a distributed system environment, as shown in **Figure 1**, it is common to find nodes that are heavily loaded with respect to nodes that are inactive or have a low

workload. An inadequate workload distribution causes an increase in the execution time of the processes which, in turn, could generate an increase in energy consumption and harm the overall performance of the system. Each node runs processes that require access to certain resources, either locally or remotely.

In summary, the scenario described consists of various nodes (devices) with processing and storage capabilities. Processes can run on different nodes and request access to resources (memory, processes, files, records, etc.) located on the same node or other nodes within the distributed system. The nodes are interconnected via a communication network (local area network, wide area network, etc.). This scenario could correspond to conventional distributed systems, distributed Internet of Things (IoT) systems, nodes in a system organized as a cluster, grid, or even cloud computing. In such cases, the proposed model would need to be adapted to the management tools of the corresponding cluster, grid, or cloud.

4. Proposed solution

To address the problem, a central Runtime (runtime software) is defined that manages the processes and shared resources and interacts with similar distributed nodes to exchange information. The central node collects information from all the nodes, applies the aggregation process and obtains the list of resource assignments to processes.

A proposal is made considering the migration of processes to achieve a better workload balancing assuming heterogeneity of the nodes (with respect to code compatibility, operating system, architecture, instruction set, etc.).

The proposed aggregation operator uses information from La Red Martínez [1], related to the current computational load of the nodes, the computational load categories and their associated weight vectors, the priorities or preferences of the processes taking into account the node state and the priorities or preferences of the processes to access the available shared resources.

The calculations related to obtaining this information, considered preliminary, are the starting point for others that will determine the processes to be migrated.

When considering aspects related to load balancing, an effective migration policy is considered, which takes into account the priority of the processes, limitations in the migration (to determine the number of processes that can pass from one node to another), the weight of resources whose state may be local with respect to the process that requires it, but could change to remote when migrating processes and the impact of this depending on the performance of the receiving node according to hardware characteristics.

An innovative variant to the aggregation operator used in La Red Martínez [1] and La Red Martínez et al. [6] will be presented, including intelligent migration of processes in heterogeneous distributed systems.

The aggregation operator is developed in six well-defined stages as shown in **Figure 2**.

Figure 2. Process migration calculation stages.

The stages correspond to the evaluation of the nodes, classifying them according to their load level (high, medium, low). Next, the processes of the nodes with high load and the resources linked to them are evaluated. Subsequently, the compatibility of the process with the possible target nodes and the node categories are evaluated according to their performance. To choose the node, priority is given to those in which the process generates less impact, taking into account the transfer weight and the performance of the possible receiving node. A list of candidate processes to be migrated is obtained.

It is important to highlight that for the evaluation and demonstration of the effectiveness of our proposed model, we have used a simulator developed within the framework of the Project: "Development of a simulator for the evaluation of classical and new algorithms for the management of shared resources in distributed systems contemplating mutual exclusion", under the code PI 126/20 of the Southern Chaco National University (Argentine). This simulator was enriched with a process migration module, which is the central focus of this article, and which was not previously present in the simulator. A series of tests and configurations have been carried out, using experimental data and scenarios that simulate real use, which has allowed us to validate and highlight the importance of our contribution in the management of shared resources in distributed systems.

In the proposed methodology, based on La Red Martínez [1], aggregation operators from the OWA (Ordered Weighted Averaging) family are used, specifically of the Neat OWA type, at each evaluation stage [37,38]. Neat OWA operators are an extension of traditional OWA operators, where the weights assigned to the input values depend on the specific characteristics of the data or the context of the application. This feature makes Neat OWA operators especially useful in applications that require precise modeling of decision-making and information fusion in complex environments.

Formal Definition:

Given a set of values $x_1, x_2, \ldots, x_n$, and a corresponding set of weights $w_1, w_2, \ldots, w_n$, where $w_i \geq 0$ and $\sum_{i=1}^{n} w_i = 1$, the OWA operator is defined as:

$$\text{OWA}\ (x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} w_i\ b_i$$

where b_i is the i-th largest value from the set $x_1, x_2, \ldots, x_n$ after sorting them in non-increasing order.

The OWA operator is often used in scenarios where the importance of each input value depends on its relative position in the ordered list rather than its absolute value.

The Neat OWA operator is an extension of the traditional OWA operator, where the weights assigned to the ordered values are not fixed but instead depend on the characteristics of the data or the context of the application. The Neat OWA operator adjusts the weights dynamically, offering more flexibility in modeling decision-making processes. This operator is particularly useful in decision-making and multi-criteria aggregation problems where the significance of each criterion varies based on the context.

The computational complexity of OWA operators is primarily determined by two main steps: sorting the input values and performing the weighted aggregation. Below is a breakdown of the complexity:

The first step is to sort the input values in non-increasing (or non-decreasing) order. This step is computationally the most expensive part and typically has a complexity of:

$O\ (n \log n)$

where n is the number of input values. This is because sorting algorithms such as quicksort or mergesort, which are commonly used, have an average time complexity of:

$O\ (n \log n)$

After sorting the values, the second step is to compute the weighted sum of the ordered values. This involves multiplying each sorted value by its corresponding weight and then summing the results. This step has a linear complexity:

$O\ (n)$

since it requires traversing the sorted list of n values and performing a constant number of operations (multiplication and summation) for each element.

The total computational complexity is dominated by the sorting step, so the overall complexity of the OWA operator is:

$$O\ (n \log n) + O\ (n) = O\ (n \log n)$$

Thus, the time complexity of computing an OWA aggregation is primarily determined by the sorting of the input values, which scales logarithmically with the number of inputs.

4.1. Nodal evaluation

The first stage of **Figure 2** consists of analyzing the scenario at the system node level, the initial values are configured, where the criteria is set to indicate the load percentages, priorities and other parameters that will be considered for the migration.

To establish the node loads, the average load of the percentage of CPU, RAM and input/output usage is calculated, which is shown in **Table 1**. Nodes whose average load is greater than 70% are considered loaded, medium load between 40% and 70%, and nodes whose average load value is less than 40% are considered as low load.

From **Table 1**, nodes with high load that can migrate processes and nodes with low load that can receive them are identified.

Table 1. Nodal loads.

Nodes	Criteria Values			Average	Load
	CPU	RAM	I/O		
1	80	90	75	81.67	High
2	30	30	50	36.67	Low
3	77	49	85	70.33	High
4	6	7	9	7.33	Low
5	5	5	6	5.33	Low
6	7	9	7	7.67	Low
7	7	10	15	10.67	Low
8	9	6	7	7.33	Low
9	10	9	15	11.33	Low
10	8	7	8	7.67	Low

Source: Own elaboration. Based on [1].

Considering that each node can host several processes and resources, it is important to mention that these processes not only compete for exclusive access to local resources (LR), but also for resources that are located on other nodes, i.e., remotely (RR). In each situation, different options that will lead to the intelligent migration of processes in the heterogeneous distributed system will be evaluated.

4.2. Process evaluation

The information resulting from the evaluation of the status of the nodes in the first stage helps to identify those with high load, their processes and the local and remote resources linked to these processes. In the second stage shown in **Figure 2**, the evaluation of processes within each node with high load begins.

The process priority criterion for all resource allocations [1] will be considered to calculate the average priority of each process.

For each resource request of each process, a priority criterion is established, which is assigned by the node's Runtime at the time the request is made. The sum of this priority value, for each requested resource, divided by the total number of requests of each process, will indicate the average priority of the process.

Processes whose average priority exceeds an arbitrarily set limit are considered candidates for migration; only those processes whose average priority exceeds 0.6 are considered. The nodes with high load are node 1 and node 3, both of which have 7 processes to be evaluated.

Node 1

average process priority (p11) = (0.8 + 0.3 + 0.9 + 0.8 + 0.95 + 0.6) /6 = 0.730

average process priority (p12) = (1 + 0.8 + 0.8 + 0.8 + 0.8 + 0.3 + 0.3 + 0.3) /6 = 0.670

average process priority (p13) = (0.6 + 0.9 + 0.9 + 0.9 + 0.5 + 0.5 + 0.8 + 0.4 + 0.9) /8 = 0.690

average process priority (p14) = (0.7 + 0.8 + 0.8 + 0.8 + 0.4 + 0.4 + 0.7 + 0.8 + 0.7) /8 = 0.660

average process priority (p15) = (0.7 + 0.8 + 0.6 + 0.8 + 0.6 + 0.6 + 0.8 + 0.5) /8 = 0.68

average process priority (p16) = (1 + 0.8 + 0.7 + 0.6 + 0.8 + 0.8 + 0.8 + 0.3 + 0.3)/8 =0.675

average process priority (p17) = (0.6 + 0.8 + 0.7) /3 = 0.675

Node 3

average process priority (p31) = (0.7 + 0.7 + 0.9) / 3 = 0.767

average process priority (p32) = (0.9 + 0.8 + 0.9 + 0.9 + 0.6) /4 = 0.800

average process priority (p33) = (0.6 + 0.3 + 0.3 + 0.8 + 0.7 + 0.6 + 0.4 + 0.6) /5 = 0.538

average process priority (p34) = (0.7 + 0.8 + 0.9 + 0.9 + 0.7 + 0.7 + 0.7 + 0.6 + 0.9) /8 = 0.775

average process priority (p35) = (0.9 + 0.9 + 0.8 + 0.6 + 0.8 + 0.4 + 0.8) /7 = 0.743

average process priority (p36) = (0.8 + 0.8 + 0.8 + 0.8 + 0.7 + 0.8 + 0.8 + 0.8 + 0.8 + 0.8 + 0.8 + 0.8) /9 = 0.789

average process priority (p37) = (0.9 + 0.5 + 0.6 + 0.8 + 0.8 + 0.8) /5 = 0.720

Process p33 is not evaluated in this cycle because for the proposed model only those processes whose average process priority exceeds the established limit, in this case 0.6, are considered.

The processes that qualify for possible migration are listed in **Table 2**.

Table 2. Processes with average priority higher than 0.6.

Nodes	Processes	Average process priority
1	p_{11}	0.73
1	p_{12}	0.67
1	p_{13}	0.69
1	p_{14}	0.66
1	p_{15}	0.68
1	p_{16}	0.66
1	p_{17}	0.70
3	p_{31}	0.77
3	p_{32}	0.80
3	p_{34}	0.78
3	p_{35}	0.74
3	p_{36}	0.79
3	p_{37}	0.72

Source: Own elaboration. Based on [1].

4.3. Compatibility assessment

In this third stage of **Figure 2**, an evaluation is performed to determine if there are compatible nodes with low load that can host processes from other nodes. The processes to be evaluated are those that passed the previous stage.

To evaluate the compatibility of the nodes, the hardware and software components (features) of the nodes are considered, which are grouped into sets. The evaluation is based on determining whether a node with low load has the same or more components than the node with high load under evaluation, by checking whether a set

of characteristics of the source node (O) is contained in another set of characteristics of a possible target node (D) or whether two sets (O and D) are the same. The formula used in this case is as follows:

Compatibility (O, D): $O \subseteq D$

The compatibility of node 1 (node with high load) is evaluated in relation to the other nodes with low load. The set of components is given by h1, h2, h3 referring to hardware components and s1, s2, s3 referring to software components.

Node 1: N1 = {h1, h3, s2}
Node 2: N2 = {h1, h2, h3, s1, s2}
Node 4: N4 = {h1, h3, s2}
Node 5: N5 = {h2, h3, s1, s3}
Node 6: N6 = {h1, s3}
Node 7: N7 = {h2, h3, s1, s2, s3}
Node 8: N8 = {h1, h3, s2}
Node 9: N9 = {h1, h2, h3, s1, s2, s3}
Node 10: N10 = {h2, h3, s1, s3}

Based on the aforementioned data sets, the following evaluation is made:

Node 2: $N1 \subseteq N2$
Node 4: $N1 \subseteq N4$
Node 5: Does not comply with the condition.
Node 6: Does not comply with the condition.
Node 7: Does not comply with the condition.
Node 8: $N1 \subseteq N8$
Node 9: $N1 \subseteq N9$
Node 10: Does not comply with the condition.

The evaluation of the hardware and software component sets of node 1 compared to the other nodes reveals that nodes 2, 4, 8 and 9 are compatible with node 1, as they contain all the features of node 1 and even some additional ones. In contrast, nodes 5, 6, 7 and 10 are not compatible due to the absence of certain required features.

The compatibility of node 3 (node with high load) is assessed in relation to the other nodes with low load.

Node 3: N3 = {h2, h3, s1, s3}
Node 2: N2 = {h1, h2, h3, s1, s2}
Node 4: N4 = {h1, h3, s2}
Node 5: N5 = {h2, h3, s1, s3}
Node 6: N6 = {h1, s3}
Node 7: N7 = {h2, h3, s1, s2, s3}
Node 8: N8 = {h1, h3, s2}
Node 9: N9 = {h1, h2, h3, s1, s2, s3}
Node 10: N10 = {h2, h3, s1, s3}

Based on the above-mentioned data sets, the following assessment is made:

Node 2: Does not comply with the condition.
Node 4: Does not comply with the condition.
Node 5: $N3 \subseteq N5$
Node 6: Does not comply with the condition.

Node 7: N3 ⊆ N7

Node 8: Does not comply with the condition.

Node 9: N3 ⊆ N9

Node 10: N3 ⊆ N10

The evaluation of the hardware and software component sets of node 3 in comparison with the other nodes reveals that nodes 5, 7, 9 and 10 are compatible with node 3, as they contain all the features of node 3 and even some additional ones. On the other hand, nodes 2, 4, 6 and 8 are not compatible due to the absence of certain required features.

For each process there can be several compatible nodes with low available load. Taking into account the set of compatible nodes with low load, the resources requested by the processes are checked to determine the candidate node for a possible migration.

4.4. Resource assessment

Once the processes of the node with high load and the possible destination nodes have been identified, the fourth stage of **Figure 2** begins, where the comparison of the local resources in relation to the remote ones is carried out, to evaluate the impact of the possible migration of processes.

Figure 3 shows the weight migration vector (WMV) where values of the migration weight that the process would generate if migrated to a certain destination node are stored. There will be a weight migration vector for each process in relation to the evaluated destination node.

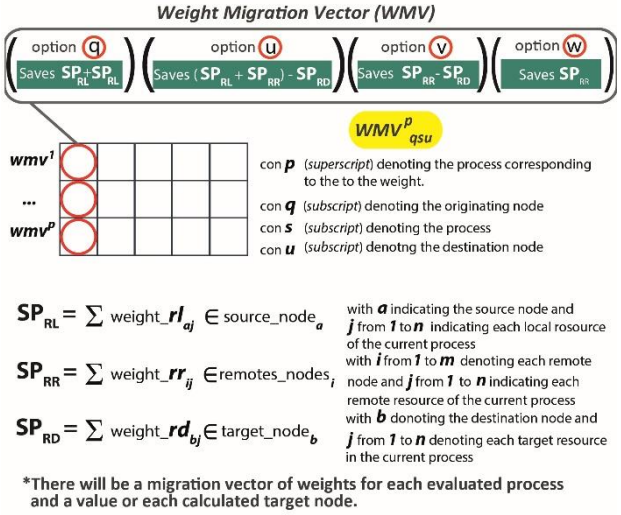

Figure 3. Weight migration vector (WMV).

For each node with high load, the evaluation of its processes is performed for the different situations that may arise. Any remote node compatible with the source node will be evaluated and may be a candidate to become a target node.

Options (*q*), (*u*), (*v*) and (*w*) in **Figure 3** are the situations that can occur for the calculation of the transfer weight in the migration of a process to another node.

Table 3 shows the process migration vector for process *p*11 which has the following associated resource weights: the local resource weight is 0.6 and the remote resource weight at node 2 is 2.1. Analyzing **Figure 3** in relation to node 2 its value is

recorded given the situation (*u*), and in relation to nodes 4, 8 and 9 its value is saved given the situation (*q*).

Table 3. Migration vector of the *p*11 process.

wmv[1]			**Options**			
q	***s***	***u***	**(*q*)**	**(*u*)**	**(*v*)**	**(*w*)**
1	1	2	-	0.6	-	-
1	1	4	2.7	-	-	-
1	1	5	-	-	-	-
1	1	6	-	-	-	-
1	1	7	-	-	-	-
1	1	8	2.7	-	-	-
1	1	9	2.7	-	-	-
1	1	10	-	-	-	-

Table 4 shows the process migration vector for process *p*12 which has the following associated resource weights: the local resource weight is 1.0, the remote resource weight is distributed over two nodes, at node 2 it is 1.2 and at node 3 it is 1.3. Process *p*12 could migrate only to node 2 as doing so to another node would generate a high impact on bandwidth.

Table 4. Migration vector of the *p*12 process.

wmv[2]			**Options**			
q	***s***	***u***	**(*q*)**	**(*u*)**	**(*v*)**	**(*w*)**
1	2	2	-	2.3	-	-
1	2	4	-	-	-	-
1	2	5	-	-	-	-
1	2	6	-	-	-	-
1	2	7	-	-	-	-
1	2	8	-	-	-	-
1	2	9	-	-	-	-
1	2	10	-	-	-	-

Table 5 shows the process migration vector for process *p*13 which has the following associated resource weights: the local resource weight is 2.1, the remote resource weight is distributed over two nodes, at node 2 it is 1.0 and at node 3 it is 2.0. Analyzing **Figure 3** in relation to node 2 its value is saved given situation (*u*). Process *p*13 could migrate only to node 2 as doing so to another node would generate a high impact on bandwidth.

Table 5. Migration vector of the *p*13 process.

wmv[3]			**Options**			
q	***s***	***u***	**(*q*)**	**(*u*)**	**(*v*)**	**(*w*)**
1	3	2	-	4.1	-	-
1	3	4	-	-	-	-
1	3	5	-	-	-	-
1	3	6	-	-	-	-
1	3	7	-	-	-	-
1	3	8	-	-	-	-
1	3	9	-	-	-	-
1	3	10	-	-	-	-

Table 6 shows the process migration vector for process *p*14 which has the following associated resource weights: the local resource weight is 2.2, the remote resource weight is distributed over two nodes, at node 2 it is 1.2 and at node 4 it is 1.9. Analyzing **Figure 3** in relation to nodes 2 and 4 their value is saved given situation (*u*). Process *p*14 could migrate only to node 2 or node 4 as doing so to another node would generate a high impact on bandwidth.

Table 6. Migration vector of the *p*14 process.

wmv[4]			**Options**			
q	***s***	***u***	**(*q*)**	**(*u*)**	**(*v*)**	**(*w*)**
1	4	2	-	4.1	-	-
1	4	4	-	3.4	-	-
1	4	5	-	-	-	-
1	4	6	-	-	-	-
1	4	7	-	-	-	-
1	4	8	-	-	-	-
1	4	9	-	-	-	-
1	4	10	-	-	-	-

Table 7 shows the process migration vector for process *p*15 which has the following associated resource weights: the remote resource weight is distributed in four nodes, in node 2 it is 0.6, in node 3 it is 1.8, in node 5 it is 2.1 and in node 6 it is 1.4. Analyzing **Figure 3** we observe at nodes 2, 8, 9, its value is recorded given the situation (*v*), and at node 4, its value is recorded given the situation (*q*).

Process *p*16 is discarded from the possibility of migration because it does not meet the conditions to migrate to other nodes as doing so would generate a high impact on bandwidth.

Table 7. Migration vector of the *p*15 process.

wmv[5]			**Options**			
q	***s***	***u***	**(*q*)**	**(*u*)**	**(*v*)**	**(*w*)**
1	5	2	-	-	5.3	-
1	5	4	5.9	-	-	-
1	5	5	-	-	-	-
1	5	6	-	-	-	-
1	5	7	-	-	-	-
1	5	8	-	-	5.9	-
1	5	9	-	-	5.9	-
1	5	10	-	-	-	-

Table 8 shows the process migration vector for process *p*17, the local resource weight of 0.9, it has no remote resources and since its weight does not exceed the limit set in situation (*p*) in **Figure 3**, any of the compatible nodes with low load of the distributed system is a candidate for possible migration and its value is saved given situation (*q*).

Table 8. Migration vector of the *p*17 process.

wmv[7]			**Options**			
q	***s***	***u***	**(*q*)**	**(*u*)**	**(*v*)**	**(*w*)**
1	7	2	0.9	-	-	-
1	7	4	0.9	-	-	-
1	7	5	-	-	-	-
1	7	6	-	-	-	-
1	7	7	-	-	-	-
1	7	8	0.9	-	-	-
1	7	9	0.9	-	-	-
1	7	10	-	-	-	-

Table 9 shows the process migration vector corresponding to process *p*31 at node 3, which is the next one with high load.

Table 9. Migration vector of the *p*31 process.

wmv[1]			**Options**			
q	***s***	***u***	**(*q*)**	**(*u*)**	**(*v*)**	**(*w*)**
3	1	2	-	-	-	-
3	1	4	-	-	-	-
3	1	5	-	-	-	2.2
3	1	6	-	-	-	-
3	1	7	-	-	0.7	-
3	1	8	-	-	-	-
3	1	9	-	-	-	2.2
3	1	10	-	-	-	2.2

Regarding process *p*31, its remote resource weight is distributed in two nodes, in node 1 it is 0.7 and in node 7 it is 1.5. Analyzing **Figure 3** and considering that process *p*31 has no local resources, in relation to nodes 5, 9, and 10, its value is recorded given the situation (*w*) and in relation to node 7 its value is recorded given the situation (*v*).

Table 10 shows the process migration vector for process *p*32 which has the following associated resource weights: the weight of remote resources is distributed in two nodes, in node 1 it is 0.9 and in node 2 it is 0.6. Analyzing **Figure 3** and considering that process *p*32 has no local resources, in relation to nodes 5, 7, 9 and 10 its value is saved given the situation (*w*).

Table 10. Migration vector of the *p*32 process.

wmv²			**Options**			
q	***s***	***u***	**(*q*)**	**(*u*)**	**(*v*)**	**(*w*)**
3	2	2	-	-	-	-
3	2	4	-	-	-	-
3	2	5	-	-	-	1.5
3	2	6	-		-	-
3	2	7	-	-	-	1.5
3	2	8	-	-	-	-
3	2	9	-	-	-	1.5
3	2	10	-	-	-	1.5

Process *p*34 is discarded from the possibility of migration because it does not meet the conditions to migrate to other nodes as doing so would generate a high impact on bandwidth.

Table 11 shows the process migration vector for process *p*35 which has the following associated resource weights: the weight of local resources is 1.5, the weight of remote resources is distributed in two nodes, in node 6 it is 1.2 and in node 7 it is 1.8. Analyzing **Figure 3** in relation to node 7, its value is saved given situation (*u*). The process *p*35 could migrate only to node 7 as doing so to another node would generate a high impact on bandwidth.

Table 11. Migration vector of the *p*35 process.

wmv⁵			**Options**			
q	***s***	***u***	**(*q*)**	**(*u*)**	**(*v*)**	**(*w*)**
3	5	2	-	-	-	-
3	5	4	-	-	-	-
3	5	5	-	-	-	-
3	5	6	-	-	-	-
3	5	7	-	2.7	-	-
3	5	8	-	-	-	-
3	5	9	-	-	-	-
3	5	10	-	-	-	-

Table 12 shows the process migration vector for process *p*36 which has the following associated resource weights: the remote resource weight is distributed over three nodes, at node 1 it is 1.8, at node 2 it is 1.6 and at node 8 it is 1.8. Analyzing **Figure 3** in relation to nodes 5, 7, 9 and 10, their value is recorded given the situation (*w*).

Table 12. Migration vector of the *p*36 process.

wmv[6]			**Options**			
q	***s***	***u***	**(*q*)**	**(*u*)**	**(*v*)**	**(*w*)**
3	6	2	-	-	-	-
3	6	4	-	-	-	-
3	6	5	-	-	-	4.6
3	6	6	-	-	-	-
3	6	7	-	-	-	4.6
3	6	8	-	-	-	-
3	6	9	-	-	-	4.6
3	6	10	-	-	-	4.6

Table 13 shows the process migration vector for process *p*37 which has the following associated resource weights, the local resource weight is 0.8, the remote resource weight is distributed over two nodes, at node 1 it is 1.1 and at node 4 it is 1.3. Analyzing **Figure 3** in relation to nodes 5, 7, 9 and 10, its value is recorded given situation (*q*).

Table 13. Migration vector of the *p*37 process.

wmv[7]			**Options**			
q	***s***	***u***	**(*q*)**	**(*u*)**	**(*v*)**	**(*w*)**
3	7	2	-	-	-	-
3	7	4	-	-	-	-
3	7	5	3.2	-	-	-
3	7	6	-	-	-	-
3	7	7	3.2	-	-	-
3	7	8	-	-	-	-
3	7	9	3.2	-	-	-
3	7	10	3.2	-	-	-

There will be a weight migrations vector (WMV) for each evaluated process and a value for each evaluated compatible destination node, these values are stored in a general migration weight matrix (GMWM) shown in **Table 14**. For each process, the set of compatible nodes with low load is recorded and for each possible evaluated destination node, the value resulting from the summation of the transfer weight of the WMV vector (obtained in **Figure 3**) is included.

Table 14. General migration weight matrix (GMWM).

Processes	Nodes							
	2	**4**	**5**	**6**	**7**	**8**	**9**	**10**
p_{11}	0.6	2.7	-	-	-	2.7	2.7	-
p_{12}	2.3	-	-	-	-	-	-	-
p_{13}	4.1	-	-	-	-	-	-	-
p_{14}	4.1	3.4	-	-	-	-	-	-
p_{15}	5.3	5.9	-	-	-	5.9	5.9	-
p_{17}	0.9	0.9	-	-	-	0.9	0.9	-
p_{31}	-	-	2.2	-	0.7	-	2.2	2.2
p_{32}	-	-	1.5	-	1.5	-	1.5	1.5
p_{35}	-	-	-	-	2.7	-	-	-
p_{36}	-	-	4.6	-	4.6	-	4.6	4.6
p_{37}	-	-	3.2	-	3.2	-	3.2	3.2

In relation to the general migration weight matrix, it is important to normalize its values. To achieve this, the proportional normalization technique is employed. In this method, each value in the matrix data set is divided by the total sum of the values, thus ensuring that the sum of the normalized values equals 1 (**Table 15**).

Table 15. General matrix of normalized migration weights (GMNMW).

Proc.	Nodes							
	2	**4**	**5**	**6**	**7**	**8**	**9**	**10**
p_{11}	0.01	0.03	-	-	-	0.03	0.03	-
p_{12}	0.02	-	-	-	-	-	-	-
p_{13}	0.04	-	-	-	-	-	-	-
p_{14}	0.04	0.04	-	-	-	-	-	-
p_{15}	0.05	0.06	-	-	-	0.06	0.06	-
p_{17}	0.01	0.01	-	-	-	0.01	0.01	-
p_{31}	-	-	0.02	-	0.01	-	0.02	0.02
p_{32}	-	-	0.02	-	0.02	-	0.02	0.02
p_{35}	-	-		-	0.03	-		
p_{36}	-	-	0.05	-	0.05	-	0.05	0.05
p_{37}	-	-	0.03	-	0.03	-	0.03	0.03

Green cells indicate that the process has some resource assigned to that node, grey cells indicate that it does not.

4.5. Establishing nodal categories

In the fifth step of **Figure 2**, the nodes are classified into different categories, considering the hardware components and the specific characteristics of each component (e.g., for the processor component: number of cores, MFLOPS, cache, etc.). **Table 16** lists the hardware components that will be evaluated, for example the components CPU (processor), RAM, and HDD (hard disk) will be evaluated, each X

represents the presence or absence of the hardware component necessary to execute the processes in each node.

Table 16. Hardware components.

Nodes	CPU	RAM	HDD
1	X	-	X
2	X	X	X
3	-	X	X
4	X	-	X
5	-	X	X
6	X	-	-
7	-	X	X
8	X	-	X
9	X	X	X
10	-	X	X

For each component being evaluated, the specific characteristics of each one will be analyzed.

In the case of the CPU component (**Table 17**), the following attributes will be analyzed: Number of cores, MFLOPS (millions of floating-point instructions per second), Architecture (Arch.), Cache (MBytes), Power consumption in watts (W).

Table 17. Attribute values of the CPU component.

Nodes	Cores	MFLOPS	Arch.	MB	W
1	8	0.01	32	12	95
2	10	0.012	64	16	110
3	8	0.009	64	10	90
4	6	0.008	32	8	80
5	4	0.006	32	6	70
6	4	0.0045	32	4	60
7	2	0.003	32	2	50
8	4	0.0075	32	6	75
9	6	0.01	64	8	85
10	2	0.0025	32	4	40

In the case of the RAM component (**Table 18**), the following attributes will be analyzed: Capacity (GBytes), Clock speed (MHz), Bandwidth (GB/s), Access time (ns), Power consumption in watts (W).

Table 18. Attribute values of the ram component.

Nodes	GB	MHz	GB/s	ns	W
1	16	3000	40	12	0.8
2	64	3200	50	10	1.2
3	128	3600	60	9	1.5
4	32	3200	45	11	1
5	64	3400	55	10	1.2
6	16	2800	35	13	0.7
7	8	2666	30	14	0.6
8	32	2800	35	11	0.9
9	256	4000	70	8	1.8
10	4	2666	32	15	0.5

In the case of the hard disk component (**Table 19**), the following attributes will be analyzed: Capacity (GB), Rotational speed (RPM), Transfer rate (MB/s), Cache (MB), Power consumption in watts (W).

Table 19. Attribute values of the hard disk component.

Nodes	GB	RPM	MB/s	Mb	W
1	2000	5400	150	128	6
2	4000	7200	250	256	7.5
3	8000	7200	300	512	8.2
4	4000	7200	200	256	6.5
5	8000	7200	250	512	7
6	500	5400	100	64	4.5
7	1000	5400	120	128	5
8	4000	5400	200	256	8.3
9	12000	10000	350	1000	9
10	1000	7200	150	128	5.5

The values of each hardware component attribute shown in **Table 16** shall be multiplied by its weight vector detailed in **Table 20** below. For each hardware component, a weight vector associated with its specific characteristics is defined, whose values may be different when it is desired to reflect the relative importance of one over the other. Without loss of generality, the same weight vector shall be used for the power calculation of each hardware component.

Table 20. Vector of weights for each hardware component.

Attribute 1	Attribute 2	Attribute 2	Attribute 4	Attribute 5
0.3	0.2	0.2	0.2	0.1

The values of the attributes of each hardware component in **Table 17** through **Table 19**, taken row by row, i.e., with respect to each node, shall be multiplied by the vector of weights in **Table 20** and then the results shall be summed to obtain the power

of each hardware component. This can be seen in **Table 21**, where each X in **Table 16** is replaced by the resulting value as the power value for each hardware component.

Table 21. Hardware component power.

Nodes	CPU	RAM	HDD
1	20.702	-	1736.200
2	30.002	671.320	2741.950
3	-	772.350	4003.220
4	17.802	-	2731.850
5	-	712.320	3993.100
6	14.401	-	-
7	-	544.460	1430.100
8	16.302	-	2372.030
9	24.702	892.580	5870.900
10	-	543.850	1796.150

The hardware component power values in **Table 21**, taken row by row, i.e., with respect to each node, will be multiplied by another vector of weights in **Table 22** and then the results will be summed to obtain the weighted index of each node's power. This can be seen in **Table 23**.

Table 22. Vector of weights for the weighted power index.

CPU	RAM	HDD
0.5	0.2	0.3

Table 23. Nodal power.

Nodes	Weighted index of each node
1	194.935
2	764.787
3	1032.349
4	207.171
5	1012.316
6	7.200
7	449.358
8	181.818
9	1454.305
10	522.385

The resulting weighted index values are classified into different categories using linguistic labels. Lower and upper limits are set for each category to which a linguistic label was assigned.

For the example three categories will be taken, sub-standard, standard and supra-standard classified according to the capabilities of each node of the distributed system

mentioned in **Table 23**. The categories are established according to the results of the node capabilities and the number of categories to be established.

According to the values in **Table 23**, the minimum value is 7,200, which will be the lower limit of the sub-standard range, and the maximum value is 1454.30, which will be the upper limit of the Supra standard range; with these values, without loss of generality, a classification into three categories has been established, the limit values between categories being the following: 489.569, 971.937.

The classification is as follows (**Table 24**).

Table 24. Classification of nodes according to their performance.

Nodes	Weighted index of each node	Classification of Performances
1	194.935	Sub standard
2	764.787	Standard
3	1032.349	Sub standard
4	207.171	Sub standard
5	1012.316	Supra standard
6	7.200	Sub standard
7	449.358	Sub standard
8	181.818	Sub standard
9	1454.305	Supra standard
10	522.385	Standard

Since the model considers the lowest handover impact, to favor high-performance nodes as receivers, the inverse of the weighted nodal power index values (**Table 23**) is considered. Their values should be normalized so that their sum is equal to 1. This can be seen in **Table 25**.

Table 25. Inverted and normalized nodal power.

Nodes	Weighted index of each node	Inverse of the weighted index	Normalized index
1	194.935	0.005	0.032
2	764.787	0.001	0.008
3	1032.349	0.001	0.006
4	207.171	0.005	0.030
5	1012.316	0.001	0.006
6	7.200	0.139	0.855
7	449.358	0.002	0.014
8	181.818	0.006	0.034
9	1454.305	0.001	0.004
10	522.385	0.002	0.012

4.6. Choice of destination node

In the sixth step of **Figure 2**, the choice of destination nodes for each process to be migrated is made. To assign the order of process migration, the general matrix of

normalized migration weights (**Table 15**) and the resulting inverted and normalized nodal power values (**Table 25**) must be evaluated.

Table 26 shows the general migration impact matrix (GMIM), resulting from the sum of the weights of the general matrix of normalized migration weights (**Table 15**) and the inverted and normalized weighted power index of each node (**Table 25**).

Table 26. General migration impact matrix (GMIM).

Proc.	**Nodes**							
	2	**4**	**5**	**6**	**7**	**8**	**9**	**10**
p_{11}	0.01	0.06	-	-	-	0.06	0.03	-
p_{12}	0.62	-	-	-	-	-	-	-
p_{13}	0.64	-	-	-	-	-	-	-
p_{14}	0.64	0.06	-	-	-	-	-	-
p_{15}	0.65	0.09	-	-	-	0.10	0.07	-
p_{17}	0.61	0.04	-	-	-	0.04	0.01	-
p_{31}	-	-	0.03	-	0.02	-	0.03	0.03
p_{32}	-	-	0.02	-	0.03	-	0.02	0.03
p_{35}	-	-	-	-	0.04	-	-	-
p_{36}	-	-	0.05	-	0.06	-	0.05	0.06
p_{37}	-	-	0.04	-	0.05	-	0.04	0.04

Green cells indicate that the process has some resource assigned to that node, grey cells indicate that it has not.

The first step to be taken from the data in the general impact matrix (**Table 26**) is to divide it into two matrices, a primary matrix (**Table 27**) (first round of allocation) and a secondary matrix (**Table 28**) (second round of allocation). The difference is that first we will try to favor those processes that have resources in the destination nodes, i.e., whose migration would imply a reduction in the input-output (transfer) impact with respect to those that do not have resources in the destination nodes; for this purpose, the primary matrix is used. The second matrix will be analyzed after having evaluated the primary matrix.

For the proposed model, each node with a high load will be able to migrate up to 45% of its processes, to avoid migrating all its processes and overloading other nodes.

Table 27. Primary matrix (PM).

Proc.	**Nodes**							
	2	**4**	**5**	**6**	**7**	**8**	**9**	**10**
p_{11}	0.01	0.06	-	-	-	0.06	0.03	-
p_{12}	0.62	-	-	-	-	-	-	-
p_{13}	0.64	-	-	-	-	-	-	-
p_{14}	0.64	0.06	-	-	-	-	-	-
p_{15}	0.65	0.09	-	-	-	0.10	0.07	-
p_{31}	-	-	0.03	-	0.02	-	0.03	0.03
p_{35}	-	-	-	-	0.04	-	-	-

Table 28. Secondary matrix (SM).

Proc.	Nodes							
	2	**4**	**5**	**6**	**7**	**8**	**9**	**10**
p_{17}	0.61	0.04	-	-	-	0.04	0.01	-
p_{32}	-	-	0.02	-	0.03	-	0.02	0.03
p_{36}	-	-	0.05	-	0.06	-	0.05	0.06
p_{37}	-	-	0.04	-	0.05	-	0.04	0.04

The lowest transfer impact in the primary matrix is considered for each process.

The choice of the destination node for the first process with the lowest transfer impact is initiated. At the end of each round, the newly assigned process and node are removed from the primary matrix. The processes whose target resources were only in the selected node become part of the secondary matrix (SM) (**Table 28**).

For each evaluation round, it is verified that the migration percentage of the source node is less than the set limit. This calculation is made on the total number of processes per node.

The evaluation continues if there are processes to be evaluated. The order of allocation of processes to target nodes of the primary matrix is indicated by the primary Allocation Function for Migration in Distributed Systems (AFMDS) (**Table 29**).

Table 29. Allocation function for migration in distributed systems (FAMSD) primary.

Process transfer weight	Selected process	Destination node
0.01	p_{11}	2
0.02	p_{31}	7
0.06	p_{14}	4

The secondary matrix is analyzed once the first round of evaluation for the primary matrix has been completed. The same evaluation criteria are applied as in the primary matrix, but for processes that do not have access requirements to resources of other nodes. The order of allocation of processes to target nodes of the secondary array is then indicated by the secondary Allocation Function for Migration in Distributed Systems (MAFDS) (**Table 30**).

Table 30. Secondary migration assignment function for distributed systems (MAFDS).

Process transfer weight	Selected process	Destination node
0.01	p_{17}	9
0.02	p_{32}	5
0.04	p_{37}	10
0.10	p_{15}	8

Processes at source nodes that have reached the migration limit are discarded and become part of a matrix of non-migratable processes (MNMP) (**Table 31**).

Table 31. Matrix of non-migratable processes (MNMP).

Proc.	Nodes							
	2	**4**	**5**	**6**	**7**	**8**	**9**	**10**
p_{12}	0.62	-	-	-	-	-	-	-
p_{13}	0.64	-	-	-	-	-	-	-
p_{35}	-	-	-	-	0.04	-	-	-
p_{36}	-	-	0.05	-	0.06	-	0.05	0.06

At the end of the evaluation of nodes 3 and 1 and having exhausted migration options or having reached the maximum percentage of process migration established, the present evaluation macro-cycle is terminated.

The primary Migration Assignment Function for Distributed Systems (MAFDS) and the secondary Migration Assignment Function for Distributed Systems (MAFDS) are then concatenated. This results in the Mapping Function for Migration in Distributed Systems Concatenated (MFMDSC) shown in **Table 32**.

Table 32. Migration in distributed systems concatenated (MFMDSC).

Process transfer weight	Selected process	Destination node
0.01	p_{11}	2
0.02	p_{31}	7
0.06	p_{14}	4
0.01	p_{17}	9
0.02	p_{32}	5
0.04	p_{37}	10
0.10	p_{15}	8

4.7. Evaluation of new overload values for inbound processes according to receiving node category

Each process has associated overhead values that indicate how much additional load it would generate on a possible receiver node in terms of memory, processor and input/output. These values are obtained using the aggregation operator developed in La Red Martínez [1].

To the average computational load obtained by the aggregation operator developed in La Red Martínez [1], an additional criterion is added, which is the overhead, obtained from the sum of the average overheads of each process in the node.

When migrating the processes, the classification table of nodes according to their performance (**Table 24**) will be considered. This will make it possible to determine the category to which both the source node and the destination node belong. In this way, the new overhead values can be calculated using the node category conversion table (**Table 33**).

Table 33. Nodal category conversion table.

Categories	**Destination**		
Source	**1**	**2**	**3**
1	1.00	0.33	0.25
2	3.00	1.00	0.75
3	4.00	1.33	1.00

Note: Category 1 corresponds to substandard, 2 corresponds to standard and 3 corresponds to supra standard.

Table 34 shows the evaluation of the new overload averages of the migrated processes according to the performance of the receiving nodes. The overhead value of the process at the origin node is multiplied by the weight value of the nodal category conversion table (**Table 33**) to obtain the new value that it will generate at the receiving node.

Table 34. Average overloads at receiving nodes.

Proc.	**Source category**	**Average origin overload**	**Destination category**	**Average target overload**
p_{14}	Standard	4.57	Standard	4.57
p_{32}	Supra standard	2.70	Supra standard	2.70
p_{31}	Supra standard	2.10	Sub standard	8.40
p_{15}	Standard	5.53	Standard	5.53
p_{17}	Standard	2.07	Supra standard	0.52
p_{37}	Supra standard	3.30	Standard	4.40

4.8. Evaluation of the impact of process migration on nodal computational load and overall system performance

The main objective is to evaluate and compare the impact of process migration on the overall state of the system. To achieve this, two stages will be carried out: in the first stage, the overload of resource allocations to processes will be considered, without performing the migration, while in the second stage, performing the migration with the new overload values according to the receiving node.

By comparing both scenarios, it will be possible to determine the impact of migration on the overall performance and operation of the system.

The computational load of each node will be calculated in two ways.

(a) Overload calculation without migration

The average overhead of the resource allocations to processes is applied to the average computational load of the **Table 1**, obtained using the aggregation operator developed in La Red Martínez [1]. This is shown in **Table 35**.

Table 35. Computational load without migration.

Node	Average	Average Overload	Final Average
1	81.67	28.13	109.80
2	36.67	11.70	48.37
3	70.33	24.27	94.60
4	7.33	0	7.33
5	5.33	0	5.33
6	7.67	0	7.67
7	10.67	0	10.67
8	7.33	0	7.33
9	11.33	0	11.33
10	7.67	0	7.67

Nodes 4 to 10 have no active processes; therefore, they have an overhead value of 0.

(b) Overload calculation with migration

The average overhead of the resource allocations to processes is applied to the average computational load of the (**Table 1**) obtained using the aggregation operator developed in La Red Martínez [1]. The average overhead values applied are the values calculated for each target node in **Table 34**. This is observed in **Table 36**.

Table 36. Computational load with migration.

Node	Average	Average Overload	Final Average
1	81.67	13.00	94.67
2	36.67	12.68	49.35
3	70.33	16.17	86.50
4	7.33	4.57	11.90
5	5.33	2.70	8.03
6	7.67	0	7.67
7	10.67	8.40	19.07
8	7.33	5.53	12.87
9	11.33	0.52	11.85
10	7.67	4.40	12.07

Node 6 has no active processes; therefore, its overload value is 0.

4.9. Evaluation of the energy consumption of the nodes once the processes have been migrated

To evaluate the energy consumption of the nodes of the distributed system, the final average overload multiplied by the corresponding value of the energy consumption conversion vector (**Table 37**) will be used. Without loss of generality, it has been considered that a node in the standard category consumes twice as much energy as one in the substandard category and that a node in the supra-standard category consumes three times as much as one in the substandard category.

Table 37. Energy consumption conversion vector.

	Categories		
	Sub standard	**Standard**	**Supra standard**
Consumption	1	2	3

The names assigned to the energy consumption categories for each node are arbitrary; a different table with more items and alternative names for each category could be used. In this case, it is assumed that the energy consumption characteristics of the nodes allow them to be grouped based on their individual consumption values into one of the three mentioned categories. Additionally, as an example, an arbitrary proportion of consumption has been established among the different categories.

The Standard category is considered representative of the average energy consumption of the nodes, the Sub-standard category corresponds to nodes with lower consumption, and the Supra-standard category is assigned to nodes with higher energy consumption.

Tables 38 and **39** show the result of the energy consumption calculation without migration and with migration.

Table 38. Energy consumption without migration.

Node	**Final Average**	**Energy Consumption**
1	109.80	109.80
2	48.37	34.71
3	94.60	283.80
4	7.33	7.33
5	5.33	16.00
6	7.67	7.67
7	10.67	10.67
8	7.33	7.33
9	11.33	34.00
10	7.67	15.33

Table 39. Energy consumption with migration.

Node	**Final Average**	**Energy Consumption**
1	94.67	63.27
2	49.35	17.95
3	86.50	55.10
4	11.90	19.50
5	8.03	23.36
6	7.67	23.73
7	19.07	12.33
8	12.87	18.53
9	11.85	19.55
10	12.07	19.33

5. Proposed metrics to evaluate load balancing and power consumption in the distributed system

To evaluate the impact of process migration and how it affects the overall state of the system, it is necessary to apply some mechanism to measure the state of the nodes before and after migration. For this purpose, the standard deviation is used, considering the final average load and energy consumption of the nodes of the distributed system.

The following criteria associated with each migrated process will be used for load evaluation: processor overload, memory overload and input/output overload.

The standard deviation of the load level of the nodes without considering migration by applying the formula is developed below.

Table 35 with the values: 109.80; 48.37; 94.60; 7.33; 5.33; 7.67; 10.67; 7.33; 11.33 and 7.67, whose average is 31.01, is considered.

$$s = \sqrt{(((109.83 - 30.95)2 + (48.37 - 30.95)2 + (94.60 - 30.95)2 + (7.33 - 30.95)2 + (5.33 - 30.95)2 + (7.67 - 30.95)2 + (10.67 - 30.95)\ 2 + (7.33 - 30.95)\ 2 + (11.33 - 30.95)\ 2 + (7.67 - 30.95)\ 2) / (10 - 1))}$$

$$s = 39.75$$

The standard deviation of the load level of the nodes with migration applying the formula is as follows.

Table 36 is considered with the values: 94.67; 49.35; 86.50; 11.90; 8.03; 7.67; 19.07; 12.87; 11.85 and 12.07, whose average is 31.40.

$$s = \sqrt{(((94.67 - 30.95)2 + (49.35 - 30.95)2 + (86.50 - 30.95)2 + (11.90 - 30.95)2 + (8.03 - 30.95)2 + (7.67 - 30.95)2 + (19.07 - 30.95)2 + (12.87 - 30.95)2 + (11.85 - 30.95)2 + (12.07 - 30.95)2) / (10 - 1))}$$

$$s = 33.50$$

The standard deviation of the energy consumption of the nodes without considering migration by applying the formula is as follows:

Table 38 is considered with the values: 109.80; 34.71; 283.80; 7.33; 16.00; 7.67; 10.67; 7.33; 34.00; 15.33; whose average is 52.66.

$$s = \sqrt{(((109.80 - 52.66)2 + (34.71 - 52.66)2 + (283.80 - 52.66)2 + (7.33 - 52.66)2 + (16.00 - 52.66)2 + (7.67 - 52.66)2 + (10.67 - 52.66)\ 2 + (7.33 - 52.66)\ 2 + (34.00 - 52.66)\ 2 + (15.33 - 52.66)\ 2) / (10 - 1))}$$

$$s = 86.92$$

The standard deviation of the energy consumption of the nodes with migration applying the formula (11) is as follows.

Table 39 is considered with the values: 94.67; 98.69; 259.50; 4.57; 3.00; 7.67; 19.07; 12.87; 35.55; 24.13, whose average is 55.97.

$$s = \sqrt{(((94.67 - 55.97)2 + (98.69 - 55.97)2 + (259.50 - 55.97)2 + (4.57 - 55.97)2 + (3.00 - 55.97)2 + (7.67 - 55.97)2 + (19.07 - 55.97)2 + (12.87 - 55.97)2 + (35.55 - 55.97)2 + (24.13 - 55.97)2) / (10 - 1))}$$

$$s = 79.71$$

From the results obtained concerning the load level of the nodes it is observed that the standard deviation of the set of final averages without migration is 39.75 and the standard deviation of the set of final averages with migration is 33.50. Referring

to the energy consumption results, it is observed that the standard deviation of the energy consumption data set without migration is 86.92 and the standard deviation of the energy consumption data set with migration is 79.71.

This indicates that the load balance and energy consumption within the distributed system has been improved because the values of the final averages dataset and the energy consumption dataset with migration are more concentrated around the mean, while the values of final averages and energy consumption dataset without migration are more dispersed.

6. Concluding remarks

This paper proposes an intelligent process migration model for the management of resources and processes in distributed systems, with the objective of solving the overload-related drawbacks that can occur in the different nodes of a distributed system. A variant of an innovative approach for the management of shared resources in distributed systems that considers workload balancing among the different nodes is presented.

The proposal is based on the evaluation of the state of computational loads, which allows detecting heavily loaded nodes with respect to inactive or low workload nodes and correcting the workload distribution in the system.

The proposed solution is based on a centralized Runtime that manages the shared processes and resources and interacts with the Runtimes of the distributed nodes to exchange information (all of them interact with their respective operating systems). In addition, initial configurations of the algorithm are established for each situation, which makes it possible to adapt the proposed solution to different environments.

Information is collected from all the nodes in a central node, the aggregation process is applied and the list of resource allocations to processes is obtained, and the additional workload that this would mean on the different distributed nodes.

The evaluation of the state of the computational loads is performed periodically, which allows considering possible migrations to balance the workload of the different nodes of the system.

As a metric to evaluate the effectiveness of the proposed solution, the standard deviation is used, which is a very useful and widely used measure of dispersion in data analysis due to its easy interpretation, sensitivity to all data values and useful mathematical properties.

The results obtained allow us to conclude that the proposed decision model and aggregation operators enable holistic management (following the integral perspective proposed in La Red Martínez [1]) of process and resource management in distributed systems. This approach also achieves workload and energy consumption balancing, while generating a precise order in resource allocation to processes that ensures mutual exclusion in accessing shared resources.

Among the significant related works evaluated and discussed in this paper, none have been found to propose a similar method for managing processes and resources in a distributed system of processing and storage nodes, achieving both workload and energy consumption balancing while also respecting distributed mutual exclusion.

As indicated, the proposed model has an acceptable computational complexity (O (n log n)), but it requires a significant supply of frequently updated data related to the resource requirements of the processes and the load state of the nodes in the distributed system.

7. Future research directions

To improve migration algorithms, we intend to research and develop more advanced and efficient algorithms for process migration that can dynamically adapt to changes in workload and distributed system conditions. This could include machine learning-based approaches for more accurate decision making.

Security policy integration is also envisioned. Research how to integrate robust security policies into the process migration process, ensuring that the migration does not compromise the integrity and confidentiality of data and processes.

Explore the application of this model in cloud environments, where resource management and process migration are key challenges, as well as evaluating its integration with container systems to improve resource management efficiency in distributed environments.

Author contributions: Conceptualization, DLLRM and FA; methodology, FA; validation, JTFM, TDMB and DLLRM; formal analysis, DLLRM; investigation, TDMB; data curation, TDMB; writing—original draft preparation, TDMB; writing—review and editing, FA; visualization, JTFM; supervision, DLLRM. All authors have read and agreed to the published version of the manuscript.

Acknowledgments: This work has been supported by the Project: “Decision models for resource and process management in distributed systems considering process migration, data imputation and fuzzy logic in new aggregation operators”, code 20F005 of the Northeastern National University (Argentine), and the Project: “Extension of a simulator for the evaluation of resource and process management algorithms in distributed systems by adding process migration”, code PI 195/24 of the Southern Chaco National University (Argentine).

Conflict of interest: The authors declare no conflict of interest.

References

1. La Red Martínez DL. Aggregation Operator for Assignment of Resources in Distributed Systems. International Journal of Advanced Computer Science and Applications. 2017; 8(10). doi: 10.14569/ijacsa.2017.081053
2. Ricart G, Agrawala AK. An optimal algorithm for mutual exclusion in computer networks. Communications of the ACM. 1981; 24(1): 9-17. doi: 10.1145/358527.358537
3. Guohong C, Singhal M. A delay-optimal quorum-based mutual exclusion algorithm for distributed systems. IEEE Transactions on Parallel and Distributed Systems. 2001; 12(12): 1256-1268. doi: 10.1109/71.970560
4. Lodha S, Kshemkalyani A. A fair distributed mutual exclusion algorithm. IEEE Transactions on Parallel and Distributed Systems. 2000; 11(6): 537-549. doi: 10.1109/71.862205
5. la Red Martínez DL, Agostini F, Acosta JC, et al. Simulator for the evaluation of algorithms for the management of shared resources in distributed systems (Spanish). Revista de Investigación en Tecnologías de la Información. 2022; 10(20): 62-79. doi: 10.36825/riti.10.20.006

6. La Red Martínez DL, Acosta JC, Agostini F. Assignment of Resources in Distributed Systems. Proceedings of IMCIC 2018-9th International Multi-Conference on Complexity, Informatics and Cybernetics.
7. Agostini F, La Red Martínez DL. Allocation of shared resources. In: Proceedings of 2019 14th Iberian Conference on Information Systems and Technologies (CISTI); 19-22 June 2019; Coimbra, Portugal.
8. Agostini F, la Red Martínez DL, Acosta JC. Assignment of Resources in Distributed Systems with Strict Consensus Requirements. Proceedings of the IMCIC 2019-10th International Multi-Conference on Complexity, Informatics and Cybernetics.
9. Agostini F. New Proposal for the Management of Resources and Processes in Distributed Systems (Spanish). Universidad Nacional del Nordeste; 2019.
10. Beiruti MA, Ganjali Y. Load Migration in Distributed SDN Controllers. NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. 2020; 1-9. doi: 10.1109/noms47738.2020.9110292
11. Upadhyay A, Lakkadwala P. Migration of over loaded process and schedule for resource utilization in Cloud Computing. In: Proceedings of 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions); 02-04 September 2015; Noida, India.
12. Deshmukh SC, Deshmukh SS. Improved load balancing for distributed file system using self acting and adaptive loading data migration process. In: Proceedings of 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions); 2-4 September 2015; Noida, India.
13. Junaidi J, Wibowo P, Yuniasri D, et al. Applied machine learning in load balancing. JUTI: Jurnal Ilmiah Teknologi Informasi. 2020; 18(2): 76. doi: 10.12962/j24068535.v18i2.a940
14. Liu H, Jin H, Xu CZ, et al. Performance and energy modeling for live migration of virtual machines. Cluster Computing. 2011; 16(2): 249-264. doi: 10.1007/s10586-011-0194-3
15. Asadi AN, Azgomi MA, Entezari-Maleki R. Analytical evaluation of resource allocation algorithms and process migration methods in virtualized systems. Sustainable Computing: Informatics and Systems. 2020; 25: 100370. doi: 10.1016/j.suscom.2019.100370
16. Chiclana F, Herrera F. & Herrera-Viedma E. Integrating Multiplicative Preference Relations in a Multipurpose Decision-Making Model Based on Fuzzy Preference Relations. Fuzzy Sets and Systems. 2001; 122(2). doi: 10.1016/S0165-0114(00)00004-X
17. Dong Y, Zhang H, Herrera-Viedma E. Consensus reaching model in the complex and dynamic MAGDM problem. Knowledge-Based Systems. 2016; 106: 206-219. doi: 10.1016/j.knosys.2016.05.046
18. Sohrabi Z, Mousavi Khaneghah E. Challenges of using live process migration in distributed exascale systems. Azerbaijan Journal of High Performance Computing. 2020; 3(2): 151-163. doi: 10.32010/26166127.2020.3.2.151.163
19. Rathore N, Chana I. Load Balancing and Job Migration Techniques in Grid: A Survey of Recent Trends. Wireless Personal Communications. 2014; 79(3): 2089-2125. doi: 10.1007/s11277-014-1975-9
20. Chang C, Hadachi A, Srirama SN. Adaptive Edge Process Migration for IoT in Heterogeneous Fog and Edge Computing Environments. International Journal of Mobile Computing and Multimedia Communications. 2020; 11(3): 1-21. doi: 10.4018/ijmcmc.2020070101
21. Marecos TD, Agostini, F, La Red Martínez D. Controlled migration of processes in distributed systems (Spanish). Proceedings of Memorias del Encuentro Argentino de Ingeniería, 6º Congreso Argentino de Ingeniería y 12º Congreso Argentino de Enseñanza de Ingeniería.
22. Fornerón Martínez JT, Agostini F, La Red Martínez DL. Resource and Process Management With a Decision Model Based on Fuzzy Logic. International Journal of Interactive Multimedia and Artificial Intelligence. 2023; 8(2): 134. doi: 10.9781/ijimai.2023.02.009
23. Bishop M, Valence M, Winiewski LF. Process migration for heterogeneous distributed systems. Dartmouth; 1995.
24. Cao J, Yu Z, Xue B. Research on collaborative edge network service migration strategy based on crowd clustering. Scientific Reports. 2024; 14(1). doi: 10.1038/s41598-024-58048-0
25. Kommisetty PDNK, Abhireddy N. Cloud Migration Strategies: Ensuring Seamless Integration and Scalability in Dynamic Business Environments. International Journal of Engineering and Computer Science. 2024; 13(04): 26146-26156. doi: 10.18535/ijecs/v13i04.4812

26. Nama P, Pattanayak S, Meka HS. AI-driven innovations in cloud computing: Transforming scalability, resource management, and predictive analytics in distributed systems. International Research Journal of Modernization in Engineering Technology and Science. 2023; 5(12): 4165-4174.
27. Devan M, Shanmugam L, Tomar M. AI-powered data migration strategies for cloud environments: Techniques, frameworks, and real-world applications. Australian Journal of Machine Learning Research & Applications. 2021; 1(2): 79-111.
28. Rathore N, Chana I. A cognitive analysis of load balancing and job migration technique in Grid. 2011 World Congress on Information and Communication Technologies. 2011; 77-82. doi: 10.1109/wict.2011.6141221
29. Hung LH, Wu CH, Tsai CH, et al. Migration-Based Load Balance of Virtual Machine Servers in Cloud Computing by Load Prediction Using Genetic-Based Methods. IEEE Access. 2021; 9: 49760-49773. doi: 10.1109/access.2021.3065170
30. Negi S, Rauthan MMS, Vaisla KS, et al. CMODLB: an efficient load balancing approach in cloud computing environment. The Journal of Supercomputing. 2021; 77(8): 8787-8839. doi: 10.1007/s11227-020-03601-7
31. Grosu D, Chronopoulos AT. A game-theoretic model and algorithm for load balancing in distributed systems. In: Proceedings of 16th International Parallel and Distributed Processing Symposium; 15-19 April 2002; Ft. Lauderdale, FL, USA.
32. Siar H, Kiani K, Chronopoulos AT. A combination of game theory and genetic algorithm for load balancing in distributed computer systems. International Journal of Advanced Intelligence Paradigms. 2017; 9(1): 82. doi: 10.1504/ijaip.2017.081181
33. Tan T, Chen X, Li C, et al. Load balancing-oriented fault-tolerant NoC design. 2024 IEEE International Test Conference in Asia (ITC-Asia). 2024; 1-6. doi: 10.1109/itc-asia62534.2024.10661350
34. He Q, Chen Y, Dong Y, et al. Efficient Load Balance Algorithm for Network-on-Chip Mapping. In: Proceedings of 2022 4th International Conference on Advances in Computer Technology, Information Science and Communications (CTISC); 22-24 April 2022; Suzhou, China.
35. Romanov A, Myachin N, Sukhov A. Fault-Tolerant Routing in Networks-on-Chip Using Self-Organizing Routing Algorithms. IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society. 2021; 1-6. doi: 10.1109/iecon48115.2021.9589829
36. Gogoi A, Ghoshal B, Manna K. Fault-aware routing approach for mesh-based Network-on-Chip architecture. Integration. 2023; 93: 102043. doi: 10.1016/j.vlsi.2023.05.007
37. Yager R. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. IEEE Transactions on Systems, Man, and Cybernetics. 1988; 18(1): 183-190. doi: 10.1109/21.87068
38. Yager R. Families of OWA operators. Fuzzy Sets and Systems. 1993; 59: 125-148. doi: 10.1016/0165-0114(93)90194-M