Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 3, pp. 336-350

Node Availability for Distributed Systems considering processor and RAM utilization for Load Balancing

A. Menendez LC, H. Benitez-Perez

Antonio Menendez Leonel de Cervantes, Hector Benitez Perez
Universidad Nacional Autonoma de Mexico
Instituto de Investigaciones en Matematicas Aplicadas y Sistemas
Departameto de Ingenieria de Sistemas Computacionales y Automatizacion
Ciudad Universitaria, Mexico D.F.
E-mail: toniomlc@gmail.com, hector@uxdea4.iimas.unam.mx

Abstract: Node-Availability is a new metric that based on processor utilization, free RAM and number of processes queued at a node, compares different workload levels of the nodes participating in a distributed system. Dynamic scheduling and Load-Balancing in distributed systems can be achieved through the Node-Availability metric as decision criterion, even without previously knowing the execution time of the processes, nor other information about them such as process communication requirements.
This paper also presents a case study which shows that the metric is feasible to implement in conjunction with a dynamic Load-Balancing algorithm, obtaining an acceptable performance.
Keywords: Node-Availability, Load Balancing, Distributed systems, High-Performance.

1 Introduction

Load-Balancing is a technique often used to distribute computational load among processors or other resources in order to get a better performance (i.e. optimal resource utilization and small processing time). When performing Load-Balancing (LB) for a Distributed System (DS) it is expected that the resources (particularly the processors) to be evenly used, therefore obtaining a general system performance increase [4]. Several studies have been carried out in terms of performance [10], task allocation [8], communication media [5], dynamic networking [1], mobile performance [5] and so on. However, these strategies depend on previous measures such as execution time of a process or communication requirements, or in standard metrics (e.g. number of processes queued at a node or processor idle time percentage) where availability (i.e. the capacity of a node to process a job) is not observed. In any case the need to measure the performance(i.e. optimal resource utilization and small processing time) of a DS with similar and almost periodic processes is not directly addressed.

In this paper a new metric named Node-Availability is introduced, it takes advantage on existing metrics such as processor and RAM utilization, the number of processes queued at a node and processes communications and compose them to create the new metric. By including several existing metrics in its calculation, Node-Availability is a metric that provides more information of a node in its value, than solely using any of the existing metrics. It compares different workload levels at two or more nodes participating in a distributed system, providing a decision criterion to be implemented in conjunction with a common workload algorithm. Dynamic scheduling and Load-Balancing in distributed systems is achieved through the Node-Availability metric, even without previously knowing the execution time of the processes, nor other information about them such as process communication requirements.

The objective of this paper is to present a metric named Node-Availability [9], how it is constructed and how it allows a DS to execute a set of processes in a balanced manner obtaining fair utilization of the overall system. One of the advantages of using the Node-Availability metric resides in its ability to

perform the load-balancing of a DS without previously knowing the execution times of the processes involved, because if the processing times where known, the scheduling and execution of the processes could be done using proven algorithms [3], [6].

The rest of this document is organized as follows: The metric is described in section 2. The LB algorithm used is presented in section 3. A case study is in section 4. The conclusions and future work are in section 5.

2 Node-Availability

The execution time of a process in a DS is a function determined by the complexity of the process, the communication strategy and by the resources available within the DS. Since this execution time cannot be easily seeing when a DS implementation is performed other strategy needs to be followed. For instance, secondary measurements such as used memory for each node or communication load amongst processors and processes can be followed. The decision of which is the most suitable measure depends entirely on the implementing resources. Considering that a Metric is a quantitative and periodic measurement interpreted in the context of a series of previous equivalent measurements, the metric to estimate the nodes availability is presented, first from the node resources point of view (2.1), followed by the tasks load (2.2) and the communications costs (2.3).

2.1 Availability

One of the most used metrics in terms of distributed computing is availability, defined as the capacity of a node to process a job at a specific time, it can be obtained from several secondary measures like the time consumed by each node (processor) or by communication performance per process.

A DS can be considered as a set of nodes communicating with each other trough a network, where node is defined as the autonomous processing unit, which consists of one processor and RAM (random access memory).

The processes that are executed in a DS generally demand to use the processor and/or memory, they are not characterized by a high input-output demand, so common measurements within nodes are the processor idle time or the percentage of free memory available. When a process demands a memory space larger than the physical RAM, the Operating System provides virtual memory to it, causing the total execution time of the process to be increased.

The percentage of processor and memory available (Figure 1) during a time sample, allows to realize what the Operating System (OS) behavior is, in terms of resource allocation to a process. The OS tries to allocate all the (RAM) memory that a process demands, it also tries to assign the processor all the possible time, to the process being executed.

Figure 1. shows a process execution which takes about 70 seconds, during this time, the operating system assigns the processor to it, resulting in a 0% processor availability. On the other hand, the RAM demand is lower than the capacity, so the availability of it is between 60% and 100%. Figure 2. shows a process demanding an amount of memory lager than the physical memory (RAM) of the node, the Operating System (OS) assigns to the process all the available RAM and then it provides virtual memory to fulfill the memory demand. As it can be seen in Figure 2. the percentage of memory available during the execution of the process is 0%, while the processor availability oscillates at the beginning and towards the end of the process execution. Figures 1 and 2 show the same process executed by the same node. It takes longer to be executed when it demands the use of virtual memory (approx. 10 seconds, Figure 2) than when it is restricted to use RAM (approx. 70 seconds, Figure 1).

The first step in our proposal is to determine the availability of a node in terms of its idle processor time and its free RAM. The highest values for these metrics indicate the most available node and the lowest values indicate the busiest node. These to metrics are multiplied for two reasons, being the first one

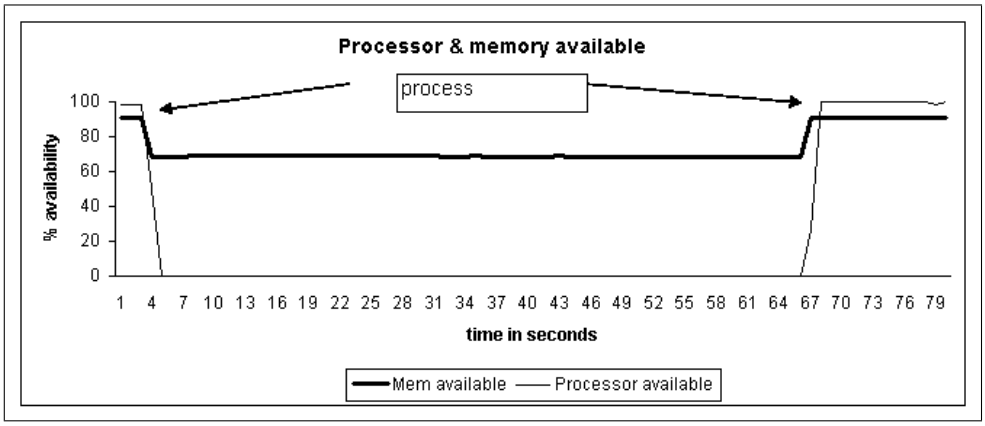

Figure 1: Memory demand within limits of RAM

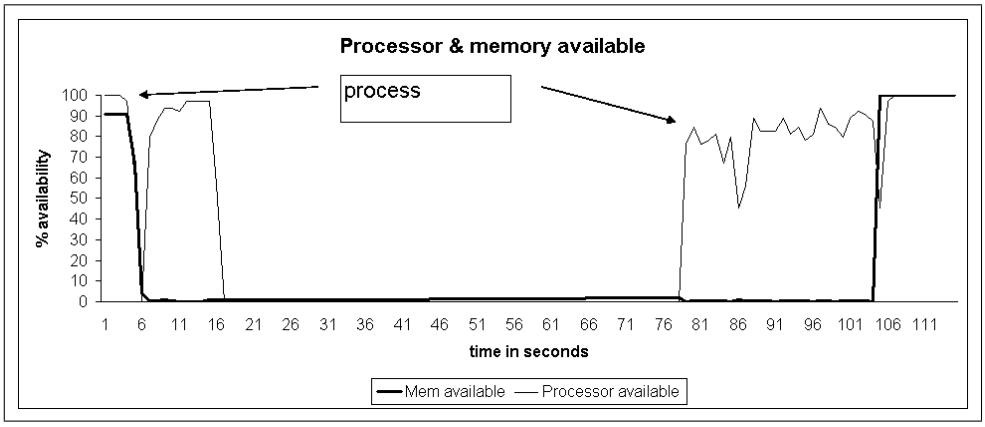

Figure 2: Demanding virtual memory

that since they are percentages the outcome of the product is also a percentage the values cannot be added because the outcome would exceed 100%. The second reason is that if the values are averaged instead of multiplied it is possible to obtain unreal outcomes, for example a node with 100% of idle processor time and 0% of available RAM averages 50%, which in terms of availability should be 0%, because if any of the two resources (processor and memory) is not available no process can be processed. On the other hand, with the proposed approach the availability is real since the value in this same example is 0%.

So the first step to evaluate the node availability of node (i) is calculated by:

$$A_i = \alpha_i \beta_i \tag{1}$$

where: $i \in \{1, 2, \ldots, n\}$ is the node identifier, α_i is the idle processor time percentage of node i, and β_i is the free RAM percentage of node i. As α_i and β_i, are percentages, they can be multiplied to find the availability of the node, giving a value between 0 and 1.

2.2 Number of processes queued at a node

The availability of a node depends not only in its respective α and β values, but also on the number of processes queued, so the previous equation of (Equation 1) must reflect this situation, considering that the number of processes affects the availability of a node in an exponential way. So Equation 1 becomes:

$$B_i = \frac{\alpha_i \beta_i}{e^{\varepsilon_i}} \tag{2}$$

where ε_i is the number of process queued at node i.

2.3 Processes communications

From the communications point of view, a process can have communications with another process in the same node that the process is being executed in, with a process that is being executed in a different node or no communications at all. The communication resources availability and the execution time is different in these three scenarios, therefore it is necessary to differentiate the processes queued in a node based on their communication requirements. So two new variables are introduced: κ_i is the number of processes queued at node i communicating (internally) with processes also queued in node i; and ν_i is the number of processes queued at node i communicating (externally) with processes queued in any node different to i.

As stated previously, the execution time of a process depends (among other things) on its communication requirements, the communication time can be approximated in terms of the type of communication (i.e. internal or external) that a process performs. As each type of communication has a different time impact, two constants are defined: ρ is the internal communication time and τ is the external communication time. The equation of Node-Availability (Equation 2) now becomes:

$$C_i = \frac{\alpha_i \beta_i}{e^{(\varepsilon_i + \rho\,\kappa_i + \tau\,\nu_i)}} \tag{3}$$

From a process communication requirements point of view, all the nodes of the DS are different. A process tends to finish its execution earlier if it communicates with processes running in the same node, therefore depending mostly on the node availability; if the processes are in different nodes, they depend not only on the different nodes availability, but also on the network speed and available bandwidth.

2.4 Number of samples taken and periodicity

Our proposal is to determine the availability of a node in terms of its idle processor time, its free RAM memory, the number of processes queued at the node and their communications. These measures are taken periodically and when δ samples have been read, then, the node availability is calculated obtaining an arithmetic average of the readings. So the final equation of Node-Availability or Φ is:

$$\Phi_{i,j} = \frac{1}{\delta} \sum_{j=1}^{\delta} \frac{\alpha_{i,j}\beta_{i,j}}{e^{(\varepsilon_{i,j}+\rho\kappa_{i,j}+\tau\nu_{i,j})}} \tag{4}$$

where: $i \in \{1,2,\ldots,n\}$ is the node identifier, $j \in \{1,2,\ldots,\delta\}$ is the sample number, $\alpha_{i,j}$ is the idle processor time percentage of node i considering the jth sample, $\beta_{i,j}$ is the free RAM memory percentage of node i considering the jth sample, $\varepsilon_{i,j}$ is the number of processes queued at node i considering the jth sample, $\kappa_{i,j}$ is the number of processes queued at node i with internal communications considering the jth sample, $\nu_{i,j}$ is the number of processes queued at node i with external communications considering the jth sample, ρ is the internal communication time, τ is the external communication time and δ is the number of samples taken of: $\alpha_{i,j}$, $\beta_{i,j}$ and $\varepsilon_{i,j}$ before sending the data to the LB process.

2.5 Optimization characteristics of Φ

In this section we show that the metric Φ presents a global minimum dependent on local values. The metric Φ is characterized per node where the discrete variables $\alpha, \beta, \varepsilon, \kappa$ and ν play the role of representing the system behavior based on the sample taken. Therefore ρ and τ are the communication time factors that can be distinguish as bounded variables and they can be delimited through a local linear optimization strategy. The time values of ρ and τ help to determine some network characteristics such as the type of protocol or the required speed specifications.

The first parameter to be defined is the Load-Balancing factor of the Distributed System(DS), which is defined as:

$$\mu_j = \frac{\Phi_{min,j}}{\Phi_{max,j}} \tag{5}$$

Where $\Phi_{min,j}$ is the Node-Availability value corresponding to the least available node and $\Phi_{max,j}$ is the value corresponding to the most available node within the DS at sample j. It can be noticed that when $\Phi_{min,j}$ and $\Phi_{max,j}$ are similar, μ_j tends to be one if the system is balanced. On the other hand, the DS is as unbalanced as μ_j tends to zero. Based upon this approximation the DS error at sample j can be defined as the DS level of unbalance Ω given by:

$$\Omega_j = \frac{(1-\mu_j)^2}{2} \tag{6}$$

To minimize this error it is necessary to balance the DS around the local loads, having that:

$$if \quad \mu_j \to 1 \quad then \quad \Omega_j \to 0$$

In order to minimize this error, the use of partial derivatives of this equation is pursued in terms of the communication times ρ and τ as shown next:

$$\frac{\partial\Omega_j}{\partial\rho_{max,j}}, \frac{\partial\Omega_j}{\partial\rho_{min,j}}, \frac{\partial\Omega_j}{\partial\tau_{max,j}}, \frac{\partial\Omega_j}{\partial\tau_{min,j}} \tag{7}$$

where: $\rho_{max,j}$ and $\tau_{max,j}$ are the communication times at the most available node, and $\rho_{min,j}$ and $\tau_{min,j}$ are the communication times at the least available node, both cases at sample j.

Now if we take into account the related node and sample values, the global error ˝ is expressed as follows:

$$\begin{aligned}\Omega_j &= \frac{1}{2}\left(\frac{\Phi_{min,j}}{\Phi_{max,j}}\right)^2 = \\ &\frac{1}{2}\left(1-\frac{\alpha_{min,j}\beta_{min,j}}{\alpha_{max,j}\beta_{max,j}}e^{(\varepsilon_{max,j}-\varepsilon_{min,j}+\rho_{max,j}\kappa_{max,j}-\rho_{min,j}\kappa_{min,j}+\tau_{max,j}\nu_{max,j}-\tau_{min,j}\nu_{min,j})}\right)^2\end{aligned} \tag{8}$$

where: *max* corresponds to the most available node, *min* corresponds to the least available node and *j* is the sample number.

Reordering this expression in terms of $\lambda_j^{(1)}$ and $\lambda_j^{(2)}$ as follows:

$$\begin{aligned}\lambda_j^{(1)} &= \frac{\alpha_{min,j}\beta_{min,j}}{\alpha_{max,j}\beta_{max,j}} \\ \lambda_j^{(2)} &= \varepsilon_{max,j}-\varepsilon_{min,j}+\rho_{max,j}\kappa_{max,j}-\rho_{min,j}\kappa_{min,j}+\tau_{max,j}\nu_{max,j}-\tau_{min,j}\nu_{min,j}\end{aligned} \tag{9}$$

Ω can be expressed as:

$$\Omega_j = \frac{1}{2}\left(1-\lambda_j^{(1)}e^{\lambda_j^{(2)}}\right)^2 \tag{10}$$

The partial derivatives are as follows:

$$\begin{aligned}\frac{\partial\Omega_j}{\partial\rho_{min,j}} &= (1-\lambda_j^{(1)}e^{\lambda_j^{(2)}})(\lambda_j^{(1)}e^{\lambda_j^{(2)}})(\kappa_{min,j}) \\ \frac{\partial\Omega_j}{\partial\rho_{max,j}} &= (1-\lambda_j^{(1)}e^{\lambda_j^{(2)}})(\lambda_j^{(1)}e^{\lambda_j^{(2)}})(-\kappa_{max,j}) \\ \frac{\partial\Omega_j}{\partial\tau_{min,j}} &= (1-\lambda_j^{(1)}e^{\lambda_j^{(2)}})(\lambda_j^{(1)}e^{\lambda_j^{(2)}})(\nu_{min,j}) \\ \frac{\partial\Omega_j}{\partial\tau_{max,j}} &= (1-\lambda_j^{(1)}e^{\lambda_j^{(2)}})(\lambda_j^{(1)}e^{\lambda_j^{(2)}})(-\nu_{max,j})\end{aligned} \tag{11}$$

The communication times ρ and τ defined in terms of the next sampling period (j+1) are expressed as:

$$\begin{aligned}\rho_{min,j+1} &= \rho_{min,j}+\eta\frac{\partial\Omega_j}{\partial\rho_{min,j}} \\ \rho_{max,j+1} &= \rho_{max,j}+\eta\frac{\partial\Omega_j}{\partial\rho_{max,j}} \\ \tau_{min,j+1} &= \tau_{min,j}+\eta\frac{\partial\Omega_j}{\partial\tau_{min,j}} \\ \tau_{max,j+1} &= \tau_{max,j}+\eta\frac{\partial\Omega_j}{\partial\tau_{max,j}}\end{aligned} \tag{12}$$

where: η is a design factor where the metric balances the performance of each node depending on its relations amongst (α, β and ε).

2.6 Metric optimization examples

In order to show the effectiveness of this technique two examples are carried out in which the metric Φ is evaluated without performing any load-balancing. The first example has a medium profile processor utilization and is called "relaxed example", the second example has high profile processor utilization and is named "restrictive example". This processor utilization is calculated with the well known metric of processor utilization by a set of periodic tasks called "Processor Utilization Factor" [16]. In both examples a set of 40 periodic tasks is evenly distributed trough 10 nodes (i.e. 4 tasks per node). The theoretical workload that a set of periodic tasks imposes to a processor can be calculated using Equation (13), in which the "Utilization" of a processor is a value under one.

$$U = \sum_{i=1}^{n} \frac{c_i}{p_i} \tag{13}$$

where: U = processor Utilization, c_i = time Consumed by the task i and p_i = Period of task i and n = number of tasks.

In both examples the purpose is to obtain the optimized values for ρ and τ that are used in the next section where the metric Φ is implemented within a load-balancing algorithm.

The setup for the two examples consists of 10 nodes with 4 periodic tasks per node, the periods and consumption times for each task differ in both cases, Table 1 shows the parameters used for the relaxed example and in Table 4 are the periods and consumption times for the tasks in the restrictive example. Notice that the first three tasks (i.e. tasks numbered 1 to 3) are identical in all the nodes, the difference in the workloads of the nodes resides in the fourth task, in which the period is modified. As it can be seen in Table 1, the fourth task at every node has a period equivalent to 8 times the node number (e.g. the period of the fourth task 4 at the node number 1 is 8, the period of task 4 in node 2 is 16 an so on).

The number of samples taken during both examples is 1000, and every 8 (i.e. $\delta = 8$) samples the value of Φ is calculated using Equation (4). The number of tasks per node with internal communications is 2 and one task has external communications.

Table 1: Tasks parameters (relaxed example)

Task Number	Period (p)	Consumption time (C)
1	8	1
2	16	2
3	32	3
4	8 times the number of node	1

The set of tasks assigned to each node according to Table1 has a processor utilization per node as listed in Table 2. This example is called "relaxed" because the maximum utilization of a processor corresponds to node number 1, and it is 0.5 (as can be seen in Table 2).

Following the optimization procedure of ρ and τ explained in the previous section, their final values (final ρ and final τ) after 1,000 samples with δ=8 are shown in Table 2.

A restrictive example is presented where the processor utilization is between 70% and 81%, as can be seen in Table 4.

The difference amongst the "relaxed" and "restrictive" example resides in duration of the periods of the first three tasks at every node, as shown in Table 3, these periods last half the time for the restrictive case, thus imposing a major workload to the processors as can be seen in Table 4. The number of samples taken is again 1000 with a δ of 8. The number of tasks with internal communications is 2 and the number of tasks with external communications is 1. As stated earlier, the main difference with the

Table 2: Processor Utilization and final values of ρ and τ per node (relaxed example)

Node	Utilization factor	Final ρ time	Final τ time
1	0.4688	0.0095	0.0127
2	0.4062	0.0935	0.0673
3	0.3854	0.1033	0.1165
4	0.375	0.1081	0.1406
5	0.3688	0.111	0.1548
6	0.3646	0.1128	0.1642
7	0.3616	0.1142	0.1708
8	0.3594	0.1152	0.1758
9	0.3576	0.1159	0.1797
10	0.3562	0.1166	0.1828

"relaxed" example resides on the "Utilization factor" for all the nodes, which is around 0.75 as shown in Table 4.

Based upon these two examples, final values pf ρ & τ represent the communication characteristics or guarantees that the DS must provide in order to have a balanced system, meaning that when a node has more processor utilization, it needs to take less time in its communications.

Table 3: Tasks parameters (restrictive example)

Task Number	Period (p)	Consumption time (C)
1	4	1
2	8	2
3	16	3
4	8 times the number of node	1

Even tough the value of τ can be seen as a local parameter, it is more common to have or guarantee a global communication time for all the nodes participating in a network, so a unique communication time for the whole network must be considered. If the value chosen for τ is the one corresponding to the minimum value for the τ's amongst all the nodes then the network is more restrictive and therefore the external communications need to be faster. On the other hand the maximum value of τ means that the communications are relaxed respect to the time they take, but the counterpart is that they not help to have a balanced DS since the least available nodes need faster external communications in order to have equivalent Node-Availability values to the most available nodes. It can clearly be noticed that the in order to provide communications at the speed required by the value of τ, the network specifications play an important role. The same reasoning applies to the values of ρ for internal communications.

The range of values for ρ and τ listed in Tables 2 for the relaxed example or in 4 for the restrictive example, indicate respectively the time that internal (ρ) and external (τ) communication must take in order to have a balanced system. Further more they show the benefits of using Φ as a metric, convenient not only to perform such a task as load-balancing, but also useful to determine the optimal duration for process communications, and in the case of τ providing the speed specifications for the communications network.

Whether to choose the minimum, maximum or average values from the restrictive or relaxed case

Table 4: Processor Utilization per node (restrictive example)

Node	Utilization factor	Final ρ time	Final τ time
1	0.8125	0.0167	0.0348
2	0.75	0.1491	0.3455
3	0.7292	0.172	0.4599
4	0.7188	0.1828	0.5138
5	0.7125	0.189	0.5452
6	0.7083	0.1931	0.5657
7	0.7054	0.196	0.5802
8	0.7031	0.1982	0.591
9	0.7014	0.1999	0.5993
10	0.7	0.2012	0.606

for ρ and τ depends strictly on the particular implementation case (i.e. network protocol and processor utilization factor).

3 The High-Low (HILO) algorithm.

In order to perform LB (load-balancing) and load distribution using the presented metric Node-Availability, a simple and well known algorithm, here named High-Low (HILO) is used. The underlying principle in HILO is to determine the most available node and the least available one. The knowledge of these nodes is used by the algorithm to perform its two main methods, the periodic method named Balance and the event triggered method named Distributor (see Figure 3). These two methods are nested depending on the arrival of new processes as shown in Figure 4, in this case the periodic Balance method is executed every period while Distributor is executed only when a new process arrives.

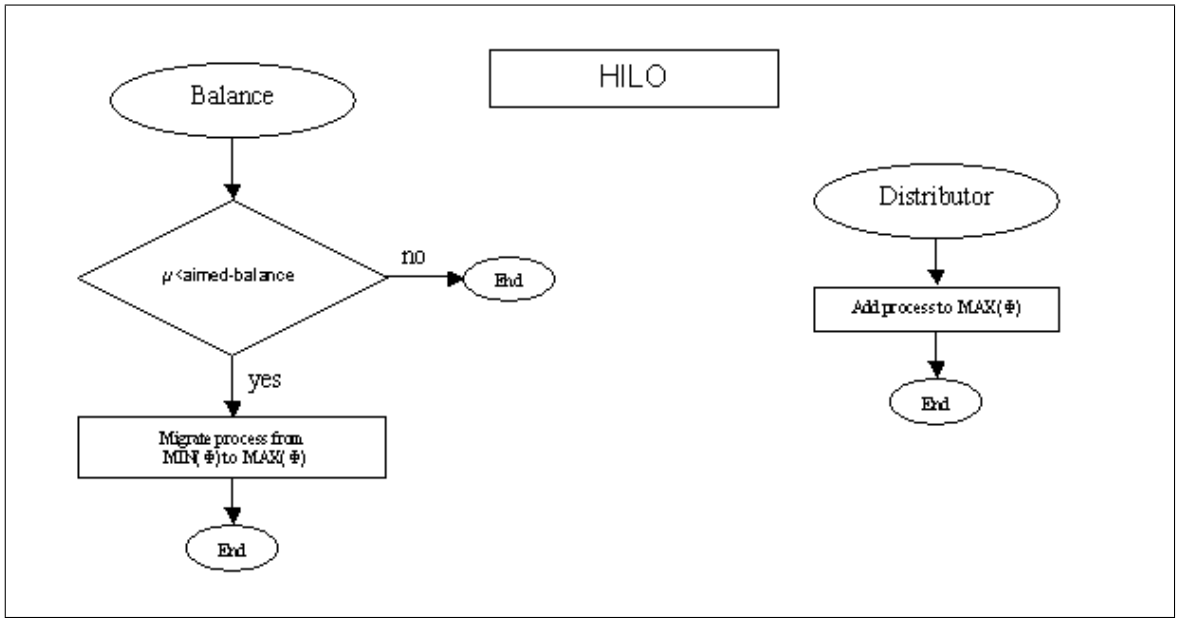

Figure 3: Flow diagram of HILO methods Balance an Distributor

The Balance method is performed as follows: the most available node is found by calculating the Node-Availability of all the nodes participating in the distributed system and selecting the one with the maximum Φ value, so the most available node is: MAX(Φ) and analogously the least available node is MIN(Φ). In order to be able to balance a DS, the HILO algorithm also requires an aimed balance level for it named ξ having $0 < \xi \leq 1$.

With this three values (MAX(Φ), MIN(Φ) and ξ), the periodical Balance method calculates the actual "Load-Balancing factor" μ using Equation (5). If the obtained value of μ is under ξ then the Balance method performs the actual load-balancing. This load-balancing is as simple as removing one process from the queue of the node MIN(Φ) and migrating it to the node named MAX(Φ).

The second method named Distributor is responsible to assign a node to any new process arriving to the distributed system. Once a new process arrives, Distributor sends it to the node MAX(Φ).

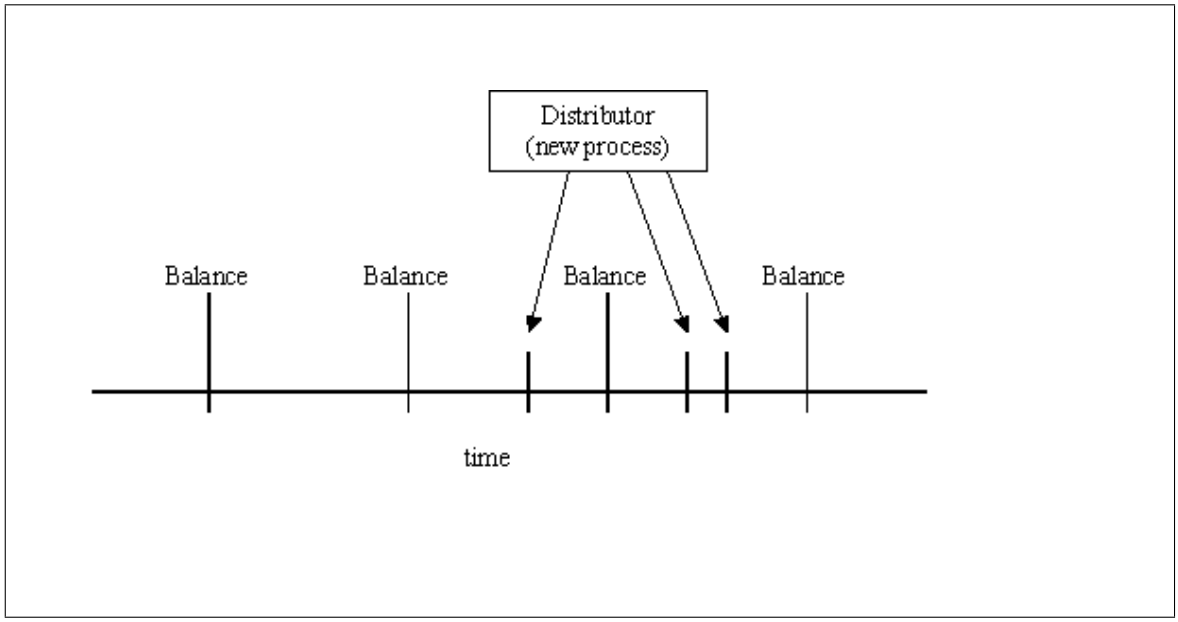

Figure 4: HILO methods: Balance an Distributor

3.1 Pseudo-code of HILO

The algorithm HILO has two methods the first one is a periodic method named "Balance", the second method which is executed every time a new process arrives to the distributed system is named "Distributor".

```
HILO
  Periodically_execute Balance
  if new Process  then Distributor
```

The Balance method:

```
Balance
 if MIN(Φ) / MAX(Φ) < ξ then
       remove Process from MIN(Φ) and send it to MAX(Φ)
```

When activated, the Distributor method sends the arriving (new) process, to the node MAX(Φ):

```
Distributor
 Add_queue MAX(Φ) new Process
```

3.2 Simulation

The HILO algorithm as well as the metric Φ are tested on both: a 16 nodes cluster (Case Study in next section) and on a simulation using Matlab. Figure 5 shows the simulation of this process where 500 samples are taken, the processes are generated between samples 50 and 250 using a Poisson distribution to simulate both; the number of processes ready to be executed and the duration of each one. It can be seeing that the system reaches an absolute balance around sample 400, but during the whole execution of the set of processes, no single node is over-occupied nor idle.

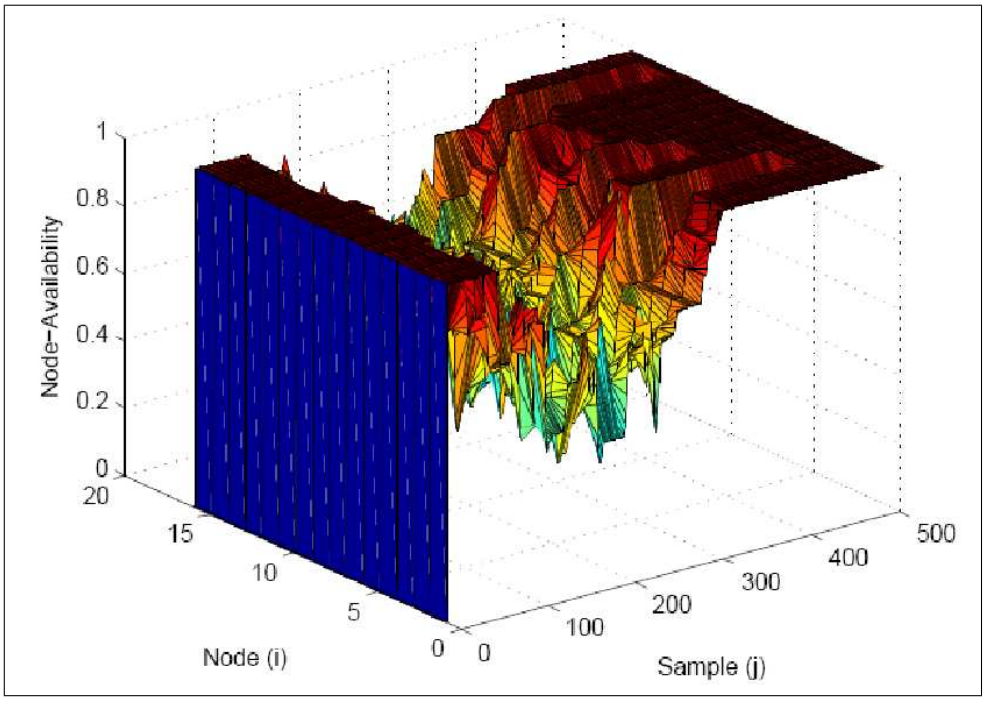

Figure 5: Simulation of a LB process within a cluster, 500 samples are taken and the processes are generated between samples 50 and 250.

The parameters used for the simulation are: Number of nodes n =16, total samples(j) = 500, and $\delta = 4$. The internal communication factor $\rho = 0.01$ and the external communication factor $\tau = 0.02$ are obtained by calculating the average times from the minimum respective values from Table 2 and Table 4. The average consumption time of each process is given by an exponential random distribution with $mean = \delta * 10$ and ξ is 0.66%. The initial value for $\alpha_{i,j}$ is one, this value is updated every time a process is queued at a node i, by decreasing its value by 0.1% per process, the approximated inverse procedure is performed when the process has finished. In terms of $\beta_{i,j}$ (available memory) a similar procedure is performed with a decrement value of 0.12%. These increments/decrements are according to the availability behavior presented in section 2.1. The value for $\varepsilon_{i,j}$ is calculated every sample, based on how many processes are queued per node and the values for $\kappa_{i,j}$ and $\nu_{i,j}$ are random numbers between 0 and $\varepsilon_{i,j}$.

The impact that Φ has on the DS load balance can be seen in Figure 5, outlining that the work load was evenly distributed amongst the 16 nodes during the whole simulation.

4 Case Study

The case study is based on a real geology-specific application, which consists of several similar processes distributed over a cluster. These processes perform a different number of operations locally. The Case study is processed seven times with a different number of processes, each occasion, as shown in Table 5. Every time the algorithm HILO and two common load distribution algorithms: Random and Round-Robin [12–14] are used to execute these 7 different sets of processes. The processes are ready to be executed based on a Poisson distribution, independent for each case study. The implementation details are presented in the 4.1 subsection, and the results in subsection 4.2.

4.1 Implementation

The case study is implemented in a dedicated cluster, which consists of 16 nodes with the following configuration:

One master node with 2 Xeon processors at 2.6 GHz, 1.5 GB RAM and Linux kernel 2.6.8. 15 nodes with Pentium IV processor at 2.6 GHz, 512 MB RAM and Linux kernel 2.6.12. The master node performs the distribution and load-balancing functions of the cluster. The case studies are integrated shown in Table 5.

Table 5: Number of processes per case study

Case study	Number of processes
1	100
2	200
3	300
4	400
5	500
6	1000
7	1500

These processes are independent amongst each other, and to simulate when a process is ready to be executed, a Poisson distribution is used. Every process performs a random number of local sums and string concatenations, both random numbers are generated globally using an exponential distribution for each case study [6], [15]. Furthermore, as both numbers differ, the demands of processor and memory are also different for every process and case study. For these cases κ and ν are equal to zero since there is no communication between processes.

Each set of processes is executed using the Random, Round-Robin and HILO algorithms to distribute the load. The Random algorithm uses a uniform distribution to select the node in which the arriving process is going to be queued. The Round-Robin algorithm sends the arriving process to the nodes in a round-robin manner. The algorithm HILO sends the arriving process to the UN node.

The algorithm HILO uses the following values for the parameters described earlier in this paper, selected (as means of example) in a heuristic manner: $\delta = 4$, $n = 15$ and $\xi = 0.6$.

4.2 Results

The total execution times of the seven sets of processes (listed in Table 1.) were obtained by executing them in the cluster, using the previously listed algorithms for the load distribution. The presented metric Node-Availability, implemented in the HILO algorithm outperforms the other two as can be seen in the

Figure 6 (Algorithm Comparison). Considering the execution time of HILO as 100%, the other two algorithms (Round-Robin and Random) take more time to complete the execution of the same seven sets of processes, this extra time goes from 10% to 65% (i.e. The execution time with Round-Robin or Random algorithms takes from 110% to 165% compared with HILO which is 100%). It can also be noticed in Figure 6 that with the smaller set of processes (i.e. 100) the percentage gain of HILO is larger, meaning that the algorithm is efficient even when the set has a relatively small number of processes.

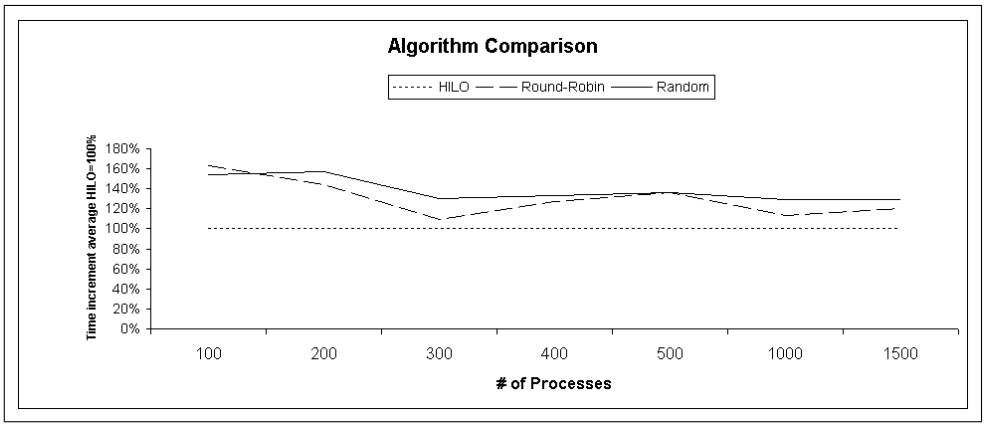

Figure 6: Percentage compared time-efficiency of the algorithm. HILO=100% .

Every occasion that the condition $\mu < \xi$ was fulfilled, a balance was performed. The total numbers of balances that HILO performed are shown in Table 6. The total execution time is expressed in minutes:seconds.

Table 6: Number of balances and total time execution per processes set.

# Processes	# Balances	Total time
100	31	01:32.0
200	86	03:17.8
300	137	04:50.0
400	186	06:27.3
500	226	07:52.0
1,000	456	15:36.0
1,500	676	22:59.9

Figure 7. shows the execution times of the processes set listed in Table5. using the "Random", "Round-Robin" and HILO algorithms, in every case HILO outperforms the other two.

5 Conclusions and future work

The metric “Node-Availability” (Φ), allows performing an efficient LB without previously knowing the execution times of the processes, nor the processes communication requirements. This metric takes into account processor and memory availability every given sample and the estimation of the related communication times per processor and process respectively.

An optimization procedure based on the communications protocol performance is carried out in order to guarantee the suitability of this metric. The time values of ρ and τ obtained after this optimization

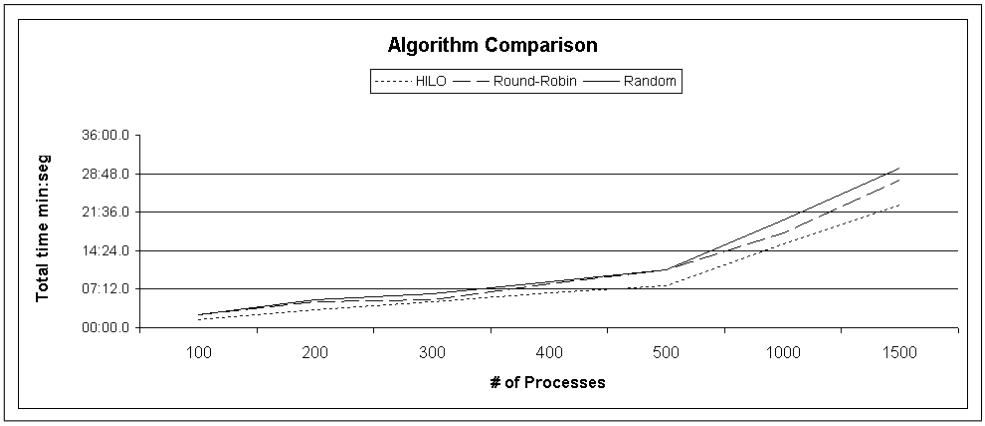

Figure 7: Total time algorithm comparison.

procedure provide the speed specifications of the communications network and the type of protocol required.

Based upon this information the proposed metric (Φ) and HILO perform an efficient response as shown in Figure 6. The results presented here provide a clear idea of the impact of Φ as the criterion metric to perform a load-balancing procedure.

For the case study taken into account, the load-balancing algorithm has a cost in terms of the time taken which can be neglected and is included in the "Total time" column of Table 6.

In terms of HILO algorithm, the ξ factor gives the possibility of balancing either in a very relaxed or strict manner. For instance, if the desired balance factor (ξ) is decreased in a significant way, the number of balances decreases towards zero. On the other hand when the value of ξ tends to 1 the system performs a load-balancing process every δ samples.

Bibliography

[1] J. Bahi, R. Couturier, F. Vernier, Synchronous distributed load balancing on dynamic networks,*Journal of Parallel and Distributed Computing*, 65 pp.1397-1405, Elsevier 2005.

[2] D. Bertsekas, *Constrained Optimization an Lagrange Multiplie Methods*, Academic Press Inc., USA 1992.

[3] J. Chiasson, Z. Tang, J. Ghanem, T. Chaouki,, J. Abdallah, D. Birdwell, M.M. Hayat, H. Jrez, The Effect of Time Delays on the Stability of Load Balancing Algorithms for Parallel Computations, *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*, VOL. 13, NO. 6, NOVEMBER 2005 pp. 932-942.

[4] R. F. de Mello, L. J. Senger, L.T. Yang, A Routing Load Balancing Policy for Grid Computing Environments, *Proceedings of the 20th International Conference on Advanced Information Networking and Applications*, 1550-445X/06 IEEE 2006.

[5] P. Ghosh, N. Roy, S.K. Das, K. Basu, A pricing strategy for job allocation in mobile grids using a non-cooperative bargaining theory framework,*Journal of Parallel and Distributed Computing*, 65 pp.1366-1383, Elsevier 2005.

[6] D. Grosu, A. Chronopoulos, Algorithmic Mechanism Design for Load Balancing in Distributed Systems,*IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICSŃPART B: CYBERNETICS*, VOL. 34, NO. 1, FEBRUARY 2004, pp. 77-84.

[7] O. Lee, M. Anshel, I. Chung, Design of an efficient load balancing algorithm on distributed networks by employing symmetric balanced incomplete block design,*IEE Proc.-Commun*, Vol. 151, No. 6, December 2004.

[8] L. Keqin, Job scheduling and processor allocation for grid computing on metacomputers, *Journal of Parallel and Distributed Computing*, 65 pp.1406-1418, Elsevier 2005.

[9] A. Menendez, H. Benitez-Perez, Node Availability for Distributed Systems considering processor and RAM utilization, *Eighth Mexican International Conference on Computer Science*, ENC07, Page(s):131 - 137, DOI:10.1109/ENC.2007.24, 2007.

[10] B. Parhami, Swapped interconection networks: Topological, performance, and robustness attributes, *Journal of Parallel and Distributed Computing*, 65 pp.1443-1452, Elsevier 2005.

[11] M. Perez, A. Sanchez, J. Pea, V. Robles, A new formalism for dynamic reconfiguration of data servers in a cluster, *Journal of Parallel and Distributed Computing*, 65 pp.1134-1145, Elsevier 2005.

[12] H. Sit, K. Ho, H. V. Leong, W. P. R. Luk, L. Ho, An Adaptive Clustering Approach to Dynamic Load Balancing, *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPANŐ04)* 1087-4089 IEEE 2004.

[13] D. Takemoto, S. Tagashira, S. Fujita, Partitioning in Content-Addressable Networks Distributed Algorithms for Balanced Zone, *Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADSŐ04)* 1521-9097 IEEE 2004.

[14] Torque Resource Manager http://www.clusterresources.com/pages/products/torque-resource-manager.php 2006.

[15] Z. Zeng, B. Veeravalli, Rate-Based and Queue-Based Dynamic Load Balancing Algorithms in Distributed Systems, *Proceedings of the Tenth International Conference on Parallel and Distributed Systems*, 1521-9097/04 IEEE 2004.

[16] Liu W.S. Jane, *Real-Time Systems*, Prentice Hall, USA, 2000.

Antonio Menéndez LC is a computer science engineer from the Universidad La Salle and currently PhD candidate by the Universidad Nacional Autónoma de México (UNAM). More than 25 years of experience in the Computer and technology industries, leading multi-million projects, as well as in the academic world. Devoted to research for the last years has participate in international congresses of real-time, convergence, hybrid systems, and Computer Science.

Héctor Benítez Pérez is a full time researcher in the IIMAS UNAM (México). He obtained his BSc in electronic engineering at the Engineering Faculty UNAM in 1994 and his PhD at Sheffield University, UK en 1999. His areas of interest are in Real Time Control and Fault Diagnosis.