Scalable Computing: Practice and Experience
Volume 10, Number 2, pp. 131–146. http://www.scpe.org
ISSN 1895-1767
© 2009 SCPE

THE IMPACT OF WORKLOAD VARIABILITY ON LOAD BALANCING ALGORITHMS[*]

MARTA BELTRÁN[†] AND ANTONIO GUZMÁN[†]

Abstract. The workload on a cluster system can be highly variable, increasing the difficulty of balancing the load across its nodes. The general rule is that high variability leads to wrong load balancing decisions taken with out-of-date information and difficult to correct in real-time during applications execution.

In this work the workload variability is studied from the perspective of the load balancing performance, focusing on the design of algorithms capable of dealing with this variability. In this paper an exhaustive analysis is presented to help users and administrators to understand if their load balancing mechanisms are sensitive to variability and to what degree. Furthermore, solutions to deal with variability in load balancing algorithms are proposed and their utilization is exemplified on a real load balancing algorithm.

Key words: cluster computing, load balancing, workload variability

1. Introduction. Cluster systems are rapidly evolving from laboratories and research centers to business environments due to their potential to take advantage of old and/or idle computers ([9]). In this kind of environment cluster systems are even more dynamic and variable than in the typical controlled research environments.

Many factors contribute to the system state variability ([5]): asynchronous scheduling and communication behavior arising from the autonomy of the individual system nodes (clusters are loosely-coupled systems), performance heterogeneity (processing nodes are rarely homogeneous, differing in many ways), workload variability (the presence of background tasks constitutes an important source of variability), variation in the number of system nodes, independent failure and recovery of nodes and variable communication latencies (network variability due to transmission errors or congestion).

Load balancing is essential to take advantage of all the available resources on cluster systems and therefore, to obtain their optimal performance ([9, 13]), especially on heterogeneous clusters. And the ability to deal with variability is essential for designing efficient and scalable load balancing algorithms. All the decisions related to load balancing are taken considering some kind of system state information. If this state suffers smooth variations, it is easy to maintain updated information about it. But when the system state is highly variable, the information may be out of date by the time the load balancing decisions are taken. Therefore, the probability of taking wrong decisions increases critically ([17]).

As far as we know the impact of workload variability on load balancing performance has not been investigated in detail. Some works have discussed similar problems in other research areas as networking ([1, 10, 11]), multimedia applications ([14]) or performance prediction ([12]), studying the variability of TCP round-trip times or another network aspects, execution times or processes arrivals respectively.

The main contribution of this paper is an exhaustive analysis of the effects of variability on load balancing algorithms performance. Furthermore, solutions for the main problems caused by these effects, related to the state information updating and to the load balancing operations, are presented. The proposed techniques can be used in any variable environment, typically a non-dedicated cluster, to obtain the desired performance with a load balancing algorithm. Two different robustness metrics have been proposed to select the best distribution rule for the algorithm, remote execution or process migration. With the proposed solution only when a high variability causes an intolerable performance degradation the process migration is used to balance the load in the system.

The rest of this paper is organized as follows. Section 2 studies the effects of workload variability on load balancing performance. To do this, load balancing algorithms are divided into four policies or rules. Section 3 proposes solutions to deal with variability in the information rule and discusses three different metrics to quantify the workload variability. Section 4 proposes solutions to deal with variability in the distribution rule and defines the robustness metric. Section 5 presents some experimental results to verify the performed analysis and to select a best variability metric. And finally, Section 6 summarizes conclusions and future work.

[*]Questions, comments, or corrections to this document may be directed to Marta Beltrán.

[†]Computer Architecture, Artificial Intelligence and Computer Science Department, Rey Juan Carlos University, Madrid, Spain ({marta.beltran, antonio.guzman}@urjc.es).

2. Effects of workload variability on load balancing. To study the effects of variability on load balancing performance, the structure of a load balancing algorithm has been decomposed in this research with the four rules or stages proposed in [19]. Another structures for the load balancing algorithm could be used to make easier the analysis of the effects of variability, but the results would be the same because it is only an structural issue and finally all the algorithms perform similar stages to balance the system load. These are the four considered rules ([8]):

- **Load Measurement:** Load balancing decisions always rely on some kind of workload or availability information of system nodes. This information is typically quantified by a load index in this first algorithm stage.
- **Information Rule:** The information policy or rule specifies how to exchange the state information between the system nodes. A good information rule should achieve a balance between incurring a low communication overhead for the state information exchange and maintaining an accurate view of the system state.
- **Initiation Rule:** This load balancing stage determines when to initiate a load balancing operation. These operations introduce a non-negligible overhead in the system; the initiation rule must weigh their cost against their expected performance benefit.
- **Load Balancing Operation:** This last rule is decomposed in three more rules: the location, distribution and selection rules.
  The location rule determines the best candidate to participate in the load balancing operation. The distribution rule specifies how to balance the load among the nodes involved in the load balancing operation. This balance can be performed by remote execution (non-preemptive allocation or initial placement of tasks) or by process migration. And finally, the selection rule chooses the load that is going to be distributed or re-distributed with this operation.

Analyzing the effects of high workload variability on these four stages, some important conclusions can be drawn.

The effects of workload variability on the first algorithm stage depend on the selected load index. If it is a good load index, it should be able to show the workload variations summarizing the node state and maintaining its stability. Therefore, this stage performance depends entirely on the selected load index, and with the proper decision, the first algorithm stage will be able to deal with variability.

On the other hand, the impact of variability on the information policy is very important. If there is a low variability this policy has an easy job because it is not difficult to maintain updated system state information in all the cluster nodes. But if the workload variability is high, the job of this policy is more difficult because it has to maintain updated state information in all the cluster nodes without causing a great network overhead with the messages generated to do it.

For the initiation policy, all the effects of variability are related to the information rule. If this rule is capable of dealing with variability and of maintaining updated state information in all the system nodes, the initiation decisions are correct and the variability does not affect this algorithm stage.

Finally, the load balancing operation performance can be affected by the workload variability in the location and distribution rules. The location rule selects the best candidate to participate in the load balancing operation. Supposing that the system state information available to decide about this aspect is updated, there is still one problem: in a highly variable environment the system state may change during this rule. The best candidate for the load balancing operation in a certain moment may be the worst immediately after.

As it has been said before, the distribution rule can be based on remote execution or on process migration. Many authors have discussed about the benefits and drawbacks of process migration for the algorithm distribution rule ([16]). Some authors have concluded that migration cannot provide better performance than other alternatives like remote execution due to its costs, while others have argued that it can significantly improve systems performance, especially when the systems are highly variable, even when its costs are high ([16]). If a task is allocated to a certain system node, and after this, the state of this node suffers a great variation, process migration allows solving this problem reallocating this task. And if the initiation rule takes a wrong decision because the state information is obsolete, the migration is again the mechanism which allows correcting this situation. The performance of algorithms based on remote execution can be affected by variability because there are not mechanisms to migrate processes once they have been assigned to one system node, even if the load balancing decisions were based on wrong state information or if the system state has changed since this assignment.

Considering this discussion, there are three main challenges that need to be faced: the load index selection, the information policy design and the load balancing operation execution. The load index selection is not included in this paper because it is a whole important research area in load balancing, but next sections try to propose solutions for the two other challenges.

3. Information rule capable of dealing with variability. Three different policies have been traditionally proposed for the information rule in load balancing algorithms:

- **Periodic (or batch) policy:** Each system node periodically broadcasts its state information through the system, regardless of whether this information is needed or not. In this kind of policy a critical question is to fix the interval between successive information exchanges, the information period (T_{info}).
- **On-demand policy:** Each system node sends a request message to the rest of the system nodes when it needs the information to make a load balancing decision.
- **Event-driven policy:** The information exchange is asynchronous, taking place when some specific conditions are met (events). These events are typically related to node state changes in a certain degree and in this case, the critical question is to decide what kind of event determine when the information exchange must be performed.

Intuitively, it can be seen that periodic approaches for the information rule provide a simple solution for information exchange, but one important problem is how to fix the interval for this exchange in highly variable environments. A long interval leads to work with obsolete information and therefore, to take load balancing decisions based on old knowledge about nodes state. But a short interval, necessary to manage updated information, implies a heavy communication overhead. The on-demand approach minimizes the number of generated messages, information is sent only when a system node requests it, introducing an operation delay. When some node needs information to evaluate a load balancing decision, it has to wait for the rest of system nodes to send it. This delays should be avoided on cluster systems, specially if the system state is highly variable and the system has a large number of nodes, because the time taken to gather all the state information is enough to make this information old. The event-driven policy might obtain a compromise between the two previous solutions, but it depends on the selected events, i. e., on the conditions which must be met to inform about the nodes state.

The event-driven policy seems to be the best alternative to obtain an information policy capable of dealing with workload variability. With this kind of policy the information exchange is asynchronous, taking place when some specific conditions are met, the events. The critical question is to decide what kind of event determine the information exchange. The proposed solution in this work is based on considering the system workload variability. If the workload of a system node is continuously changing (large variability values), this node must inform very often about its state, but if its workload hardly never changes (low variability values), this information exchange is unnecessary.

Let V denote the workload variability (independently of the metric selected to quantify it, discussed in the next section) and $V_{\max}$ denote its maximum value. The information events are defined in the following way:

$$\frac{V}{V_{\max}} > \chi. \tag{3.1}$$

With this event definition a system node informs the rest of the system nodes about its state when its workload variability is above a certain fraction of its maximum value. The normalization allows to consider system heterogeneity, if one node workload changes a fraction of 0.5 of its maximum value, the meaning is the same for all the system nodes with independence of this maximum value. Events are defined with the 'relative variability' not with absolute values which cannot be compared in different system nodes with different features. The cluster user (or administrator) only has to set the threshold value, χ, used to determine how updated must be the global state information in the system nodes.

3.1. Variability quantification. In this section, different ways of quantifying the system workload variability are proposed to define the information rule events. In this context, the variability metric must quantify how much the workload data differ from individual to individual on a system node. Each node maintains a vector (X) with its last K load index values. The load index quantifies the workload (or the availability, inversely related to this load) of the node at a given instant. Load balancing algorithms use to measure this load index periodically on each node to maintain updated information about system state, therefore the vector X keeps the index values measured in the last K monitoring intervals. Taking into consideration these values, there are three main alternatives to measure the workload variability ([4]):

- **The range (R):** This metric is obtained with the difference between the largest measurement in a set (the maximum) and the smallest (the minimum).

$$R = \max(X) - \min(X). \tag{3.2}$$

- **The inter-quartile range (IQR):** Another kind of range not sensitive to the extreme values or to the sample size is the inter-quartile range. This metric is the difference between the first and third quartiles of the sample ($Q1$ and $Q3$ respectively).

$$IQR = Q3(X) - Q1(X). \tag{3.3}$$

  Note that $Q1$ is the $25th$ percentile, therefore, the value below which a 25% of observations fall. And $Q3$ is the $75th$ percentile, the value below which a 75% of observations fall.
- **The standard deviation (σ):** This is a well-known variability metric and it is a kind of average of the distances of the observed values from their mean ($\overline{X}$). If many of the observations are far above or below the mean, the deviation value is large.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{K} (X_i - \overline{X})^2}{K-1}}. \tag{3.4}$$

It is important to be able to interpret these two lasts metrics meaning, a little more complex than the range. And to understand that their values cannot be compared due to the differences in their meanings. With the IQR definition the variability can be interpreted as the dispersion of the values in the center of the sample, not taking into account the smallest 25% (smaller than the first quartile, $Q1$) and the largest 25% (larger than the third quartile $Q3$): the IQR is expected to include about half of the data. On the other hand, the standard deviation is a measure of how spread out or close together the sample values are, quantifying the average distance from their mean value.

4. Distribution rule capable of dealing with variability. First, to avoid the negative effects of workload variability on the location rule, negotiated operations should be always used.

Once a system node performs the initiation rule and decides to initiate a load balancing operation, the location rule selects the best system node to be involved in the load balancing operation. Before finishing this rule, the node initiating the operation always asks the the system node selected to perform the load balancing operation to accept this operation. This node can accept or reject the load balancing operation depending on its own state information, always more updated than its state information in the rest of nodes. Therefore, if the selected node has changed its state during the initiation and location rules due to a high variability, and its new workload is higher, it can reject the operation and the location rule has to be performed again.

For the distribution rule, the desirable solution should obtain the costs of remote execution, but having some mechanism to redistribute the allocated tasks in highly variable environments. Thus, the load balancing algorithm should rely on a non-preemptive allocation for the load distribution. This kind of policy is easier to implement and implies a lower cost to the system than a preemptive one ([16]). But, is it enough to maintain the desired system performance in all the possible environments, even in highly variable situations? The answer to this question is not.

In this work a remote execution rule is proposed as the general distribution mechanism, but using a robustness metric to decide if an exceptional migration is needed due to a high variability. The robustness metric is proposed to quantify the degradation of the load balancing algorithm performance when large system state variations take place. This degradation quantification allows the user to decide if process migration is needed to obtain the desired performance with the load balancing algorithm when large state changes occur. These large variations caused by the workload variability will be called perturbations from this point.

A tasks assignment is defined to be robust with respect to a system performance feature against some specific perturbations if the degradation in this feature is limited when the perturbations occurs ([3]). But, what is the degree of this assignment robustness in a dynamic load balancing algorithm? A system user or administrator may need this information to tune properly the algorithm parameters and to be sure of not needing process migration to correct in real time the tasks allocation to keep the desired system performance when perturbations occur.

Siegel proposed the FePIA procedure ([2, 3]) to quantify a system-task allocation combination robustness. This method, very frequently used for analyzing systems robustness in initial task assignments, constitutes the starting point which will be extended in this paper to quantify the robustness of a dynamic load balancing algorithm. Therefore, when the process allocation performed by the algorithm is robust enough against perturbations, process migration is not needed. On the other hand, if these perturbations caused by variability lead to an unacceptable performance degradation, the process migration must be used to redistribute the tasks among the system nodes.

With the inclusion of this robustness criteria, the distribution rule of the load balancing algorithm can deal with variability but without incurring in unnecessary overheads and costs in low variable environments.

Two different robustness metrics have been defined: the QoS-robustness and the P-robustness. The first metric should be used when the load balancing algorithm objective is related to the individual tasks composing the global application executed on the cluster and the second should be used with global algorithm objectives related to the total application.

4.1. QoS-robustness. This robustness metric should be used when the load balancing algorithm decisions are taken to achieve an objective related to the tasks composing the application executed on the cluster. The initiation mechanism evaluates the tasks response times or their resources utilization or their balance to decide when to begin a new load balancing operation.

In this case the performance feature is a quality-of-service value (QoS) related to the individual tasks: response time, resources utilization, balance, etc. The system is robust when:

$$A \geq \tau_R \cdot \overline{A}, \tag{4.1}$$

where A denotes the real QoS achievement obtained when the application is executed, $\overline{A}$ denotes its expected achievement and τ_R is the robustness tolerance defined by the system designer or user.

Therefore, the load balancing algorithm tries to obtain a certain QoS value related to the application tasks and the assignment made by the algorithm is robust if it obtains a certain degree of the expected QoS, given by τ_R, despite all the possible perturbations in the system.

The achievement value (A) has to be defined to quantify the degree of the QoS achievement, therefore:

$$A = \frac{W_C}{W} \tag{4.2}$$

where W_C is the number of tasks executed on the system accomplishing with the expected QoS and W is the total number of tasks composing the application.

Following the FePIA procedure, the steps proposed to quantify the proposed robustness are:

1. **Performance Features (Ψ):** The selected performance feature must be related to the QoS achievement:

$$\Psi = \{\psi_j\} = \{w_j | 1 \leq j \leq N\},$$

where w_j is the number of tasks allocated to the j_{th} system node that finally achieve the expected QoS and N is the number of system nodes. There is an equivalent magnitude, $\overline{w_j}$, denoting the total number of tasks allocated to this node that are supposed to achieve the expected QoS if there are not system perturbations to avoid it.

2. **Perturbation Parameters (Π):** The parameters whose values can impact the proposed performance feature must be identified. System perturbations may modify the system state preventing tasks to accomplish with the expected QoS. The perturbation parameter is then:

$$\Pi = L,$$

where L denotes the total number of tasks that are allocated to system nodes to achieve the expected QoS but finally are not able to do it due to perturbations. Therefore, $\overline{L} = 0$, because the system expects not to fail any prediction about the achievement of the QoS requirements.

3. **Impact of the perturbation parameter on the system performance feature:** With the previous definitions, this relation can be summarized with the following expressions:

$$W_C = \sum_{j=1}^{N} w_j, \qquad \overline{W_C} = \sum_{j=1}^{N} \overline{w_j}, \qquad W_C = \overline{W_C} - L.$$

4. **Robustness metric:** It is the smallest variation in the value of the perturbation parameter that would cause the performance feature to violate its acceptable variation, i. e. a violation of the condition expressed in the equation 4.1. And the load balancing algorithm is not robust if this condition is not fulfilled. The degree of the QoS achievement can be related to the performance feature and to the perturbation parameter by:

$$A = \frac{W_C}{W} = \frac{\sum_{j=1}^{N} w_j}{W} = \frac{\overline{W_C} - L}{W},$$

and, therefore, the robustness radius is:

$$\rho_{QoS}(\Psi, \Pi) = \min_{L:A(L) \geq \tau_R \cdot \overline{A} \wedge \exists L+1:A(L+1) < \tau_R \cdot \overline{A}} (L). \tag{4.3}$$

Therefore, the robustness radius is a discrete magnitude: the largest number of tasks executed on the system that can finally fail in achieving the expected QoS without causing an unacceptable performance degradation in terms of the robustness tolerance given by the system designer or user.

This procedure allows to algorithmically evaluate the discrete robustness radius of the system and to use this value to decide about the need of process migration. The robustness metric warns the designer or user when the system-load balancing algorithm combination is not able to obtain the expected performance in terms of expected QoS achievement (A). This warning can be obtained in two different situations:

- Before executing the application if there is a priori information about it and about the system.
- After executing the application, once this information has been collected, to improve the performance for later executions.

4.2. P-Robustness. When the load balancing algorithm tries to achieve a global objective for the total application (total response time, global resources utilization, system balance, etc) this second robustness metric based on the global system power should be used. In [15], the following system power definition P is proposed:

$$P = \lambda \cdot f\left(QoS, \overline{QoS}\right), \tag{4.4}$$

where λ is the system throughput (finished tasks per time unit) , QoS is the quality of service metric chosen for the evaluated system and $\overline{QoS}$ is the expected or desired quality of service.

The three general QoS metrics proposed in this work are the application response time (T), the system efficiency (ε) and the system balance (B):

- The function f proposed in [15] for the response time metric can be easily used in this work:

$$f\left(T, \overline{T}\right) = \frac{1}{1 + \frac{T}{\overline{T}}}. \tag{4.5}$$

  With this definition, the system power is high when its throughput is high and when the T value is close to its expected value, $\overline{T}$.
- For the efficiency and balance metrics, a new f function has been proposed in this work. Since a high response time value and a high efficiency or balance values have opposite effects on the power figure, this function for the efficiency and balance is:

$$f\left(\varepsilon, \overline{\varepsilon}\right) = \frac{1}{1 + \frac{\overline{\varepsilon}}{\varepsilon}}, \quad f\left(B, \overline{B}\right) = \frac{1}{1 + \frac{\overline{B}}{B}}. \tag{4.6}$$

Therefore, for this robustness metric the performance feature whose degradation is quantified is the whole system power P. In this case, the system is robust when

$$P \geq \tau_R \cdot \overline{P}, \tag{4.7}$$

where P denotes the real system power (equation 4.4) when the application is executed on the cluster, $\overline{P}$ denotes its expected value and τ_R is the robustness tolerance defined by the system designer or user again.

Therefore, the system is robust if it obtains a certain degree of the expected system power, given by τ_R, despite all the possible perturbations in the system.

The steps to obtain the system robustness in this case are the following :

1. **Performance Features (Ψ):** The selected performance feature is the finishing time of each system node:

$$\Psi = \{\psi_j\} = \{M_j | 1 \leq j \leq N\},$$

where M_j is the time at which the j_{th} system node finishes executing all its allocated tasks, $\overline{M_j}$ is its expected value and N is the number of system nodes. The total application response time will be related to this magnitude and to the starting time of the application (T_o):

$$T = \max(M_j) - T_o.$$

Because the last node finishing its work is the one limiting the total application response time.

2. **Perturbation Parameters (Π):** The parameters whose values can impact the proposed performance feature are the processes response times. Therefore the perturbation parameters are continuous in this case:

$$\Pi = \{\pi_i\} = \{t_i | 1 \leq i \leq W\},$$

where t_i denotes the response time of the i_{th} task on the system and again $\overline{t_i}$ is the expected response time for this task.
But with the previous definition, the following steps in the procedure are very difficult to complete, because the current cluster nodes are almost always multitasking systems. Therefore the perturbation parameter is slightly transformed in:

$$\Pi = \{\pi_i\} = \{F_i | 1 \leq i \leq W\},$$

where F_i denotes the time at which the i_{th} task finishes its execution and $\overline{F_i}$ denotes its expected value.

3. **Impact of the perturbation parameter on the system performance feature:**
This impact can be summarized with this expression:

$$M_j = \max(F(q_j(k))) \text{ with } 1 \leq k \leq |q_j|,$$

where F is a vector composed of all the F_i values (for $1 \leq i \leq W$) and q_j is a vector whose indices denote the tasks allocated to the j_{th} system node. For example, if $q_1 = [1\ 5\ 8]$, the cluster node number 1 has been allocated three tasks, whose finishing times are F(1), F(5) and F(8) and $M_1 = \max(F(1), F(5), F(8))$.

4. **Robustness metric:** Again, it is the smallest variation in the value of the perturbation parameter that would cause the performance feature to violate its acceptable variation, i. e. a violation of the condition expressed in the equation 4.7.

$$P \geq \tau_R \cdot \overline{P}. \tag{4.8}$$

Therefore, given the system power definition:

$$\lambda \geq \tau_R \cdot \overline{P} \cdot f^{-1}(QoS, \overline{QoS}) \geq \tau_R \cdot \overline{P} \cdot \overline{f^{-1}(QoS, \overline{QoS})},$$

then

$$\lambda \geq \tau_R \cdot \overline{\lambda}.$$

From this point and taking into account the traditional throughput definition ($\lambda = W/T$):

$$\frac{W}{T} \geq \tau_R \cdot \frac{W}{\overline{\overline{T}}}, \qquad \frac{1}{T} \geq \tau_R \cdot \frac{1}{\overline{\overline{T}}}, \qquad T < \frac{1}{\tau_R} \cdot \overline{T}.$$

where T is the application response time, and if T_o is its starting time:

$$T = \max(M_j) - T_o, \quad \overline{T} = \max(\overline{M_j}) - T_o = \max(\overline{F_i}) - To,$$

and therefore:

$$\max(M_j) - T_o < \frac{1}{\tau_R} \cdot (\max(\overline{F_i}) - T_o), \quad \max(M_j) < \frac{1}{\tau_R} \cdot (\max(\overline{F_i}) - T_o) + T_o.$$

And if the last node in finishing its work must fulfill this condition, all the system nodes must do it:

$$M_j < \frac{\max(\overline{F_i})}{\tau_R} - \left(\frac{1-\tau_R}{\tau_R}\right) \cdot T_o.$$

The robustness radius of M_j against F is then:

$$r_j(\Psi, \Pi) = \min_{T:M_j=\frac{\max(\overline{F_i})}{\tau_R}-\left(\frac{1-\tau_R}{\tau_R}\right)\cdot T_o} ||F - \overline{F}||_2. \tag{4.9}$$

Hence, for the entire system the robustness radius for the P-robustness will be:

$$\rho_P(\Psi, \Pi) = \min(r_j(\Psi, \Pi)). \tag{4.10}$$

Equation 4.9 is the Euclidean distance between any vector composed of the real task finishing times and the vector composed of the expected finishing times. It can be interpreted as the distance from a point to an hyperplane, therefore, using the point to plane expression, this equation can be reduced to:

$$\rho_P(\Psi, \Pi) = \min\left(\frac{\frac{\max(\overline{F_i})}{\tau_R} - \left(\frac{1-\tau_R}{\tau_R}\right) \cdot T_o - M_j(\overline{F})}{\sqrt{|q_j|}}\right). \tag{4.11}$$

With $M_j(\overline{F}) = \max(\overline{F(q_j(k))})$, and $1 \leq k \leq |q_j|$.

5. Experimental Results. To exemplify the application of the proposed solutions, a load balancing algorithm is implemented in this section. With this algorithm and a simple application, very interesting conclusions can be drawn about the utility of the solutions proposed in the previous sections and about the best metric to quantify the wokload variability. Only some examples of the performed experiments are shown in this section, but all the obtained results with different algorithms, applications and cluster systems are considered for conclusions, to make them general and meaningful in most of the typical contexts.

5.1. Example of load balancing algorithm. The following four rules have been implemented to perform the experiments:

- **Load measurement:** In the example algorithm the following new load index has been defined:

$$I = \frac{C \cdot \alpha}{C_{\max}}. \tag{5.1}$$

It can be seen that the proposed index is based on two main parameters, C and α. The first one is a static parameter, the node computing power (C). To compute the index value a normalization is performed on this magnitude, using the computing power of the fastest system node ($C_{\max}$). Therefore the $C/C_{\max}$ ratio gives some kind of 'relative computing power' of the system node. The computing power is a static feature of the system nodes, it can be quantified in the simplest approach using the *bogomips* value given by the Linux operating systems for each system node, or using more sophisticated and complex metrics such as the inverse of the response time of a certain task or process, etc.
The second parameter is a dynamic one, the CPU assignment (α). The node CPU assignment is a dynamic feature containing its availability information and it is defined as the percentage of CPU time that would be available to a new process ([6, 18]). For example, if a 30% of the CPU is available for a a new task, this task should be able to obtain the 30% of the processor time slices, and therefore $\alpha = 0.3$.

With this definition, the proposed index I is inversely related to the node workload: the higher the I value, the less loaded the node. It is an availability index instead of a load index.
Selecting the correct monitoring interval, this index has demonstrated to follow the system state variations (even in highly variable environments) due to the utilization of the dynamic parameter α, which can be computed or predicted with different models. Furthermore, it is a stable index and a good indicator of processes response times on system nodes.

- **Information rule:** Following the rules given in section 3 to deal with variability , the information rule is event-driven with the events defined in equation 3.1. The variability metric and the number of values considered to calculate it (K) can be configured by the user.
- **Initiation mechanism:** To derive the initiation rule for the considered algorithm, the methodology proposed in [7] has been used. This procedure is based on describing qualitatively the requirements for the load balancing algorithm and on identifying an objective function to quantify the achievement of these requirements.
The proposed initiation mechanism is sender initiated, the initiation rule of an overloaded node decides about load balancing operations trying to achieve the selected objective. For our algorithm, two different objectives have been implemented, one related to the individual tasks composing the executed application and another to the complete application.
The two implemented initiation mechanism are based on boundary functions (using this kind of objective function the overhead caused by optimizations is avoided). The implemented objective functions define an upper or lower bound to a certain system parameter and this condition can be directly transferred to the initiation mechanism in the algorithm.
  - **Task response time objective:**
  In this first version of the algorithm implemented to perform the experiments, the aim is to bound the individual processes response time to finish a certain number of tasks per time unit.
  Therefore this first objective is related to the individual tasks response times. The initiation bound is referred to the response time of a new task in its *home node*, that is, the node to which this task is initially assigned. Let $\hat{t_i}$ be the response time of the i_{th} task in its home node, t_i the response time of the i_{th} task regardless it finally runs, and τ be the algorithm tolerance used to define the threshold of the selected objective. This objective function implies that the initiation rule tries to allocate new tasks to nodes where their response times will not be more than τ times greater than in their home nodes:
  $$t_i < \tau \cdot \hat{t_i} \;\; \forall i.$$
  In the home node the objective function is achieved if the CPU assignment for the new task is:
  $$\alpha = \frac{1}{\tau}.$$
  This CPU assignment is the minimum necessary to accomplish the response time requirements for the new task on this node, but due to system heterogeneity, this may not be the minimum in another node (for example, on a node with more computing power). The objective function is based on the load indexes values, which take into account the nodes computing powers. The minimum index necessary to achieve the objective is:
  $$I_{\min} = \alpha_{\min} \cdot \frac{C_{\text{home}}}{C_{\max}} = \frac{1}{\tau} \cdot \frac{C_{\text{home}}}{C_{\max}},$$
  where C_{home} is the computing power of the home node and $C_{\max}$ is the computing power of the most powerful system node. And the initiation mechanism is:
  $$I > \frac{1}{\tau} \cdot \frac{C_{\text{home}}}{C_{\max}}. \tag{5.2}$$
  With this initiation mechanism the load balancing algorithm must always look for a node which fulfill this bound to allocate the new task.

– **Total response time objective:**
This second objective is related to the total application response time. In this case the user or administrator determines a bound to the finishing time of the application (T_F):

$$T_F < \tau.$$

Therefore $M_j < \tau \ \forall j$, and: $M_j = \max(t_i^o + t_i)$.
With t_i^o the arriving time for the i_{th} task to the j_{th} node and t_i is the response time of the i_{th} task. The response time predicted for this task is related to the CPU assignment it receives in the node it is allocated:

$$t_i = \frac{t_i^{\text{unloaded}}}{\alpha}.$$

With t_i^{unloaded} the response time of the i_{th} task on the j_{th} node when its processor is unloaded and α the CPU assignment that this node can offer to this new task.
Therefore, the initiation mechanism for this objective is:

$$t_i^o + \frac{t_i^{\text{unloaded}}}{\alpha} < \tau. \tag{5.3}$$

This initiation mechanism is always evaluated on the task *home node*, so this node must have information to evaluate this bound for the rest of system nodes. Each node must know the unloaded CPU response time of the tasks composing the application (t_i^o) and the computing power of the system nodes (C_j). Another important data is the average time consumed to initiate a load balancing operation ($\overline{t_{LB}}$). With all this information, the initiation mechanism of the load balancing algorithm must look for a node which fulfill this bound to allocate the new task:

$$t_i^o(\text{home}) + \overline{t_{LB}} + \frac{t_i^{\text{unloaded}}(\text{home}) \cdot \frac{C_{\text{home}}}{C_j}}{\alpha_j} < \tau. \tag{5.4}$$

It must be pointed that $t_i^o(\text{home}) + \overline{t_{LB}}$ would be the starting time of the new tasks in the receiver node, considering the average time needed to initiate the load balancing operation. The $\overline{t_{LB}}$ is the same for all the possible receiver nodes and depends mainly on the cluster network. The only case in which $\overline{t_{LB}} = 0$ is when a local execution is performed. On the other hand, the rest of values needed to perform the initiation mechanism are well known static parameters except α_j which can be computed from the last index value (I) received from the j_{th} system node.

- **Load balance operation**: Once a load balancing operation is initiated, the load balancing algorithm tries to allocate the new task accomplishing with the algorithm objective using the location rule.
Negotiated operations are always performed, thus, the sender node always asks the receiver one to accept the task and the receiver node can accept or reject the load balancing operation.
The proposed algorithm is based on non-preemptive allocation, and only newly arriving tasks can be selected to perform load balancing operations in the selection rule. Only when the robustness metric recommends it, process migration can be used.

5.2. Application and Experimental Setup. A real heterogeneous cluster called Medusa (Table 5.1) has been used to perform the desired experiments. It is a cluster composed by 32 different nodes interconnected with a Gigabit Ethernet network.

The application executed on this cluster system to evaluate the solutions proposed in this work is composed of different kinds of independent tasks with different CPU requirements.

The CPU-bound task programmed for the experiments ($t100$) performs a simple image processing operation the 100% of its response time, i. e., it can always be running on the processor. The IO-bound tasks compute this operation too, but they have to read from and write to the hard disk (IO operation) a certain fraction of their total response time. For example, $t10$ is a task which is a 10% of its response time processing the image (using the CPU) and the rest of the time performing disk accesses. The task $t20$ is performing the image processing a 20% of its response time, and so on.

Table 5.1
Cluster Medusa (Q=32)

Nodes	Processor	GHz	RAM (MB)	Bogomips
n0-n17	Pentium IV	1.7	256	3394.76
n18-n20	Pentium III	0.864	128	1723.59
n21-n31	Pentium III	0.664	128	1327.10

Table 5.2
Results for the information rule in experiment 1 (low variability)

Information policy	T_{info}/χ	T (s)	S
Periodic	1	345	1.26
Periodic	3	318	1.37
Periodic	10	302	1.44
Periodic	20	275	1.59
Periodic	30	310	1.41
Periodic	60	336	1.30
Periodic	120	549	0.79
Periodic	180	805	0.54
Periodic	300	1098	0.40
Periodic	600	1416	0.31
On-demand	0	232	1.88
Events	0.1	235	1.86
Events	0.3	229	1.90
Events	0.4	218	2.00
Events	0.5	284	1.54
Events	0.7	293	1.49
Events	0.8	327	1.33

In the next sections, the results of three different experiments will be shown, experiment 1 with low variability (dedicated cluster, only executing the example application), experiment 2 with medium variability (non-dedicated cluster with 3 random load peaks during the experiment in system nodes from n0 to n11) and experiment 3 with large variability (non-dedicated cluster with 6 random load peaks during the experiment in all the system nodes). The unpredictable load peaks have been simulated with a set of short tasks arriving to the nodes every 0.3 seconds and consuming CPU and memory resources.

The example application is composed by 320 tasks (W=320), all arriving to the lower half of the system (nodes from n0 to n15) and the elapsed time of this application without load balancing algorithm is $t_{without\ algorithm}$= 436 seconds in the experiment 1, $t_{without\ algorithm}$= 498 seconds in the experiment 2 and $t_{without\ algorithm}$= 545 seconds in the experiment 3.

5.3. Experimental results for the information rule. In the first set of experiments the three traditional information policies have been compared in environments with different variabilities. The second version of the load balancing algorithm has been used, therefore the initiation mechanism tries to allocate tasks to maintain the total response time below a certain value. For the event driven policy, the IQR metric with K=5 has been used to quantify the workload variability.

Tables 5.2, 5.3 and 5.4 show the results obtained for experiments 1, 2 and 3. In all these tables the total application response time (T) and the speedup obtained comparing this time with the response time without load balancing algorithm (S) are shown. For the periodic and event-driven policies different values of the information period (T_{info}) and of the event threshold (χ) have been used.

It can be seen that in the low variability environment (experiment 1) the best speedup results are obtained with the event-driven policy, specifically with a threshold value of χ=0.4 the speedup obtained with the algorithm is 2.00. But the on-demand and periodic policies are able to achieve very similar results, in the case of the periodic approach, with an information interval of T_{info}=20 s, the speedup value is 1.59, and for the on-demand policy the speedup is 1.88.

When the environment variability increases, two phenomenons can be observed. The first, to obtain the best speedup values the information exchange must occur more frequently. Therefore, for the periodic approach the information period must decrease and the same must happen with the event threshold for the event-driven

Table 5.3
Results for the information rule in experiment 2 (medium variability)

Information policy	T_{info}/χ	T (s)	S
Periodic	1	377	1.32
Periodic	3	354	1.41
Periodic	10	395	1.26
Periodic	20	416	1.20
Periodic	30	459	1.08
Periodic	60	481	1.04
Periodic	120	699	0.71
Periodic	180	993	0.50
Periodic	300	1344	0.37
Periodic	600	1854	0.27
On-demand	0	342	1.46
Events	0.1	276	1.80
Events	0.3	255	1.95
Events	0.4	288	1.73
Events	0.5	376	1.32
Events	0.7	397	1.25
Events	0.8	412	1.21

Table 5.4
Results for the information rule in experiment 3 (large variability)

Information policy	T_{info}/χ	T (s)	S
Periodic	1	423	1.29
Periodic	3	461	1.18
Periodic	10	488	1.12
Periodic	20	501	1.09
Periodic	30	548	0.99
Periodic	60	599	0.91
Periodic	120	625	0.87
Periodic	180	889	0.61
Periodic	300	1245	0.44
Periodic	600	3874	0.14
On-demand	0	417	1.31
Events	0.1	293	1.86
Events	0.3	360	1.51
Events	0.4	408	1.34
Events	0.5	456	1.20
Events	0.7	515	1.06
Events	0.8	539	1.01

policy. These best values can be obtained with χ=0.3 and T_{info}= 3 s for the experiment 2 and with χ=0.1 and T_{info}= 1 s for the experiment 3.

The second, the only information policy capable of achieving similar results in the highly variable system than in the low variability environment is the event-driven policy once the χ value is adequately tuned. The speedup decreases only from 2.00 to 1.86 when the variability changes, although the worsening for the on-demand policy is from 1.88 to 1.31 and for the periodic policy is from 1.59 to 1.29.

Once the event-driven has been selected to deal with variability, the second set of experiments has been performed to determine the best metric to quantify the workload variability (V) for load balancing purposes. Table 5.5 shows the obtained results using different variability metrics in the experiment 3, the most significative due to its large variability value. Different χ values for the events definition and different lengths (K) for the vector with the last load index values have been used. The number of information events (and therefore, the number of information messages, directly related to the network overhead), and the total application response time have been measured to evaluate the load balancing algorithm performance, E and T respectively.

In a low variability environment the results obtained with the three metrics would have been very similar with the K value properly configured. In a stable situation it is not necessary to inform very often about the nodes state and with a few events is enough to have updated information about the global state and to take

Table 5.5
Variability metrics comparison in experiment 3 (large variability)

Event	χ	K	E	T (s)	S
Range	0.1	3	130	341	1.60
Range	0.3	3	104	412	1.32
Range	0.5	3	71	521	1.05
Range	0.7	3	49	596	0.91
Range	0.1	5	141	350	1.56
Range	0.3	5	117	415	1.31
Range	0.5	5	95	533	1.02
Range	0.7	5	63	590	0.92
IQR	0.1	5	92	293	1.86
IQR	0.3	5	69	360	1.51
IQR	0.5	5	31	456	1.20
IQR	0.7	5	17	515	1.06
IQR	0.1	10	101	308	1.77
IQR	0.3	10	73	385	1.42
IQR	0.5	10	45	478	1.14
IQR	0.7	10	28	543	1.00
σ	0.1	5	84	306	1.78
σ	0.3	5	58	374	1.46
σ	0.5	5	25	470	1.16
σ	0.7	5	8	526	1.04
σ	0.1	10	81	328	1.66
σ	0.3	10	52	399	1.37
σ	0.5	10	19	502	1.09
σ	0.7	10	7	588	0.93

correct load balancing decisions. The range metric can obtain good results in this kind of situation, because there are not important differences between successive load index values. Anyway, this metric may have better performance with low K values, because the higher this value is the more probability of having measurement errors affecting the variability value (the range increases with the size of the data set as it has been said in section 3.1). On the other hand, the IQR metric can obtain very good results in these environments, even better than the range, because it ignores these extreme values. And finally, the standard deviation is able to obtain similar results than the other two metrics with low variability values.

But the results showed in Table 5.5 are quite different because in this case the system is suffering a high variability. The χ value which leads to the best response times is very low in all the cases because there are a lot of state changes in the system nodes and therefore, more events and information messages are needed to maintain updated state information and to make correct load balancing decisions.

As it was expected, the range metric performance is very poor in this kind of environment. Even with a low K value, (K=3) the effects of the extreme values on the metric value turns it into a bad election.

However, the IQR metric avoids these problems because it ignores the extreme values. And, furthermore, it can react to unforeseen load changes in a short interval of time, enough to maintain up-to-date state information in almost all the environments. Table 5.5 shows that this metric can obtain the best response time value with an acceptable number of events.

On the other hand, the standard deviation metric suffers the consequences of smoothing the state changes. Its slow variation with the load index changes increases the response time and decreases this metric performance in environments with high variability, because the load balancing decisions are very often based on obsolete state information. The load index changes take longer to have effects on the variability values with this metric than with the range or with the IQR, due to the use of the data arithmetic mean in its calculation and this leads to a low number of events and to a poor information updating.

To conclude, it can be seen that the best results have been obtained with K=5. With the application used in these example experiments, 5 load index values are enough to measure the variability, while larger values suppose a too long-term approximation. But it has been observed that with other applications, the best value for K could be another one. Therefore, the optimal value for this parameter is application-dependent and must be tuned properly depending on the application executed on the cluster.

5.4. Experimental results for the load balancing operation. In these last experiments, the convenience of process migration in experiments 1, 2 and 3 has been evaluated using the robustness metrics derived in section 4. In these three experiments the two versions of the load balancing algorithm with the two proposed objectives have been used.

With all the equations proposed in section 4 to perform the robustness analysis, the robustness radius for different system configurations can be theoretically computed.

First, the robustness radius for the load balancing based on the task response time objective can be computed with the QoS-robustness, because it is an objective related to the individual tasks composing the application. The robustness tolerance has been fixed in τ_R=0.85 and the expected achievement is $\overline{A}$=1, therefore:

$$A \geq 0.85 \cdot \overline{A} = 0.85.$$

The application executed in these experiments is composed of 320 tasks, therefore:

$$A = \frac{W_C}{W} = \frac{W_C}{320} \geq 0.85, \quad W_C \geq 320 \cdot 0.85 = 272,$$

and:

$$W_C = \overline{W_C} - L \geq 272, \quad L \leq 48.$$

The robustness radius for this system-application combination is 48 using the QoS-robustness. On the other hand, if the total response time objective is considered, the P-robustness radius must be computed, because it is a global objective related to the total application. With the same robustness tolerance, the following condition must be fulfilled:

$$P \geq 0.85 \cdot \overline{P}.$$

In this case, to compute the expected system power:

$$P = \lambda \cdot \frac{1}{1+\frac{T}{\overline{T}}}, \quad \overline{P} = \overline{\lambda}\frac{1}{1+\frac{T}{\overline{T}}} = \frac{320}{\overline{T}} \cdot 0.5.$$

Because in the ideal situation $T = \overline{T}$. In the example application the expected response time is $\overline{T}$=300 s. Therefore:

$$\overline{P} = 0.53.$$

And the P-robustness condition is in this case:

$$P \geq 0.85 \cdot 0.53 = 0.45.$$

Knowing the desired finishing time for all the tasks composing the application, the vector $\overline{F}$ can be built. And with this vector and the rest of available information once the application have been executed on the system one time (supposing that there is not a priori information), the robustness radius for all the system nodes can be computed. In this case the minimum robustness radius is the one corresponding to the node $n2$. For this node the vector q is:

$$q_2 = [2\ 7\ 21\ 34\ 58\ 63\ 94\ 103\ 120\ 193\ 206\ 315].$$

This vector stores the identifiers of all the application tasks allocated to this node, therefore, this node has executed 12 tasks ($|q_2| = 12$) . Using the expression 4.11 for this node and supposing the starting time $T_o = 0$:

$$\rho_2 = \frac{\frac{300}{0.85} - 281}{\sqrt{12}} = 20.76\ s$$

with

$$M_2(\overline{F}) = \max(\overline{F(q_2(k))}) = F(206) = 281\ s.$$

TABLE 5.6
QoS-robustness for experiments 1, 2 and 3

Experiment	L	Kind of system
1	14	QoS-robust
2	20	QoS-robust
3	33	QoS-robust

TABLE 5.7
P-robustness for experiments 1, 2 and 3

Experiment	$\max(\lvert F(k) - \overline{F(k)}\rvert)$	Kind of system
1	6.45	P-robust
2	15.67	P-robust
3	22.45	No P-robust

The meaning of this results is very simple to interpret. In the case of the QoS-robustness, only 48 tasks or less can fail in achieving the objective of having a response time below $\overline{t}$ to comply with the robustness condition given in equation 4.1. It has to be pointed that it is a discrete radius because it is a number of tasks.

On the other hand, for the P-robustness the robustness radius is continuous because it is a time measurement. The meaning of this radius value is the maximum deviation a task can have from its expected finishing time without causing a violation of the condition given in equation 4.7. Therefore, the maximum difference $|F(k) - \overline{F(k)}|$ should be 20.76 s to have a robust system-application combination.

To validate these conclusions, an experimental analysis has been performed. The obtained results are shown in Tables 5.6 and 5.7 for the QoS-robustness and for the P-robustness respectively. Real executions of the studied application have been performed on the Medusa cluster to measure the average L values (number of tasks that are executed without achieving the desired response time $\overline{t}$), the objective achievement value (A), the maximum deviations of the tasks finishing times from their ideal finishing times($\max(|F(k) - \overline{F(k)}|)$), the application response time (T) and the system power (P) in experiments 1, 2 and 3. And the conditions expressed in equations 4.1 and 4.7 have been evaluated to verify if the predictions made with the robustness radius about the system robustness are correct.

The results shown in Tables 5.6 and 5.7 verify the proposed robustness metrics and the expressions proposed to compute their associated robustness radius. The L value is always below the QoS robustness radius of 48 and therefore the system is always able to fulfill the robustness condition expressed in 4.1. The same thing occurs with the P-robustness in experiments 1 and 2. But on the other hand, the condition expressed in equation 4.7 cannot be fulfilled for the experiment 3. Therefore, the $\max(|F(k) - \overline{F(k)}|)$ value is above the P-robustness for this experiment and process migration would be needed to achieve the desired system robustness.

6. Conclusions and Future Work. A deep analysis of the effects of variability on load balancing performance on cluster systems has been presented. This analysis has demonstrated that these effects are very important for the information rule and the load balancing operation (assuming a good load index selection).

The main contribution of this paper is the proposal of specific solutions for considering the workload variability in this algorithm stage without causing unnecessary overheads on the system nodes or network. These solutions allow to improve the load balancing algorithms performance in highly variable environments such as non dedicated clusters.

For the information rule, an event-driven solution is proposed. The events must be determined with the load index variability on the system nodes. Experimental results have demonstrated that the best variability metric for load balancing purposes is the inter-quartile range (IQR) in all the possible environments. The number of load index values considered for its calculation (K) and the threshold for the information events (χ) must be configured to optimize the load balancing performance depending on the degree of variability and on the executed application.

For the location rule negotiated load balancing operations should be used to give the selected node the chance to reject the operation if its state has changed during the load balancing decisions.

And finally, for the distribution rule, a non preemptive allocation is proposed. But a robustness metric must be used to determine if process migration is needed in certain variable situations to maintain the desired system performance. Two new robustness metrics, the QoS-robustness and the P-robustness have been proposed for individual tasks algorithm objectives and for global objectives, respectively.

Both robustness metrics have demonstrated their accuracy in warning the user or administrator when process migration should be used to manage the workload variability because it is causing intolerable performance degradations. A very interesting line for future research is incorporating the robustness computation to the load balancing algorithm to decide in real-time which distribution rule should be used depending on the workload variability: remote execution or process migration.

REFERENCES

[1] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay, *Variability in TCP round-trip times*, in Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, ACM Press, 2003, pp. 279–284.
[2] S. Ali, A. Maciejewski, H. Siegel, and J.-K. Ki, *Definition of a robustness metric for resource allocation*, in Proceedings of the International Symposium on Parallel and Distributed Processing, 2003, pp. 42–50.
[3] S. Ali, H. Siegel, and A. Maciejewski, *The robustness of resource allocation in parallel and distributed computing systems*, in Proceedings of the Third International Symposium on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004, pp. 2–10.
[4] T. Anderson and J. D. Finn, *The new statistical analysis of data*, Springer, 1997.
[5] R. J. Anthony, *Self-configuration in parallel processing*, in Proceedings of the 16th International Workshop on Database and Expert Systems Applications, 2005.
[6] M. Beltran and J. L. Bosque, *Estimating a workstation CPU assignment with the DYPAP monitor*, in Proceedings of the Third International Symposium on Parallel and Distributed Computing, 2004, pp. 64–70.
[7] M. Beltrán, J. L. Bosque, and A. Guzmán, *Initiating load balancing operations*, in Proceedings of the EuroPar2005, 2005, pp. 292–301.
[8] M. Beltrán, A. Guzmán, and J. L. Bosque, *Dealing with heterogeneity in load balancing algorithms*, in Proceedings of the Fifth International Symposium on Parallel and Distributed Computing, 2006, pp. 123–132.
[9] R. Buyya, *High Performance Cluster Computing: Architecture and Systems, Volume I*, Prentice-Hall, 1999.
[10] K.-T. Chen, P. Huang, C.-Y. Huang, and C.-L. Lei, *The impact of network variabilities on TCP clocking schemes*, in Proceedings of the 4th Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 4, 2005, pp. 2770–2775.
[11] J. J. Evans and C. S. Hood, *Network performance variabilty in NOW clusters*, in Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, 2005, pp. 1047–1054.
[12] R. Gusella, *Characterizing the variability of arrival processes with indexes of dispersion*, IEEE Journal on Selected Areas in Communications, 9 (1991), pp. 203–211.
[13] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 2003.
[14] C. Hughes, P. Kaul, S. Adve, R. Jain, C. Park, and J. Srinivasan, *Variability in the execution of multimedia applications and implications for architecture*, in Proceedings of the 28th Annual International Symposium on Computer Architecture, vol. 4, 2001, pp. 245–265.
[15] P. Jogalekar and M. Woodside, *Evaluating the scalability of distributed systems*, IEEE Transactions on Parallel and Distributed Systems, 11 (2000), pp. 589–603.
[16] D. S. Milojicic, F. Douglis, D. Paindaveine, M. Wheeler, and Z. Zhou, *Process migration*, ACM Computing Surveys, 32 (2000), pp. 241–299.
[17] R. Mirchandaney, D. Towsley, and J. A. Stankovic, *Analysis of the effects od delays on load sharing*, IEEE Transactions on Computers, 38 (1989), pp. 1513–1525.
[18] R. Wolski, N. Spring, and J. Hayes, *Predicting the CPU availability of time-shared unix systems on the computational grid*, in Proceedings of the 8th International Symposium on High Performance Distributed Computing, IEEE, 1999, pp. 105–112.
[19] C. Xu and F. C. M. Lau, *Load Balancing in Parallel Computers: Theory and Practice*, Kluwer Academic Publishers, 1997.

Edited by: Marek Tudruj, Dana Petcu
Received: January 22th, 2009
Accepted: June 28th, 2009