# On the Dynamic Load-Rebalancing of BSP Application Using Process Migration

Rodrigo da Rosa Righi, Alexandre Carissimi, Philippe O. A. Navaux
Institute of Informatics - Federal University of Rio Grande do Sul - Porto Alegre, Brazil
{rodrigo.righi, asc, navaux}@inf.ufrgs.br

## Abstract

*We have developed a model for dynamic process scheduling in heterogeneous and non-dedicated environments. This model acts over a BSP (Bulk Synchronous Parallelism) application, applying load-rebalancing through migration of BSP processes to new processors. A BSP application is divided in one or more supersteps, each one containing both processing and communication phases followed by a barrier synchronization. In this context, the developed model combines three metrics, Memory, Processing and Communication, in order to measure the migration capacity of each BSP process. The final idea is to offer a mathematical formalism involving these metrics, and to decide the following questions about the process migration: When? Where? Which? This paper presents our load-rebalancing model, the parallel machine organization, some implementation issues and related work.*

## 1. Introduction

BSP (Bulk Synchronous Parallelism)[3] applications are composed by a set of independent processes that execute supersteps. Each superstep is subdivided into three phases: (i) local computations in each process; (ii) global communication actions; (iii) a barrier synchronization. The BSP model does not specify how the processes should be mapped to resources and the programmer must deal with this issue of scheduling in order to achieve better application efficiency. This issue is important once the barrier phase waits for the slowest process (process that spent more time in processing and communication phases) to start the next superstep. This topic is more relevant when it is considered an heterogeneous (different processor and network equipment speeds), a dynamic (fluctuations in network bandwidth and processor load) and non-dedicated distributed environment.

The allocation scheme of BSP processes to resources can be onerous to the programmer, once he must know the parallel machine architecture and utilization as well as the application writing properly. In addition, each new BSP application requires another effort for processes scheduling. A possibility is to explore the scheduling using automatic load balancing at middleware level, linking it to the BSP programming library. For example, a processor allocation scheme where the processes with larger computing times are mapped to faster machines can be used. However, this approach is not the best one for dynamic applications and distributed environments, because a good process-resource mapping performed when the process will be launched can compromise application performance[5]. At this moment, it is not possible to recognize neither the amount of processing of each BSP process nor communication patterns among them. An alternative is to perform BSP processes rescheduling through their migration to new resources, offering load-rebalancing of BSP processes in application runtime.

In this context, this paper describes a scheduling model that offers dynamic and runtime load-rebalancing over a BSP application, controlling the remapping of processes to different resources. Its parallel machine aims to join the power of clusters, LAN networks and multiprocessor machines, and presents the concept of hierarchy in two levels using sets (considering the network level) and set managers. The resultant machine can be heterogeneous regarding the processors and the networks speeds and is used in a non-dedicated manner. The final idea of the model is to adjust the conclusion of both local processing and global communication phases of each BSP process to be as fast as possible profiting from information collected at runtime. This adjust happens through the migration of those processes which have a large processing time, perform several communication actions with other processes that belong to the same set and present a low network cost. This paper shows the developed model, the implementation issues and related works.

## 2. Load-Rebalancing Model of BSP Processes

BSP load-rebalancing model acts over both computation and communication phases of each process. The idea of this model is to minimize the application execution time and improve the environment resources utilization through the cooperation of each BSP process. The load-rebalancing model controls the runtime migration of BSP processes and is launched automatically following a specified time interval and is totally transparent to the program-

mer. The final result is a mathematical formalism that answers the specified questions regarding processes migration: (i) "When" to launch the mechanism for processes migration; (ii) "Which" processes must be migrated; (iii) and "Where" to put an elected process for migration.

## 2.1. Model of Parallel Machine

The load-rebalancing model works over a heterogeneous, non-dedicated and dynamic distributed environment. The heterogeneous issue considers the processors speed (all processors have the same architecture) and network speed and level (Fast and Gigabit Ethernet and multi-clusters environment, for instance). The non-dedicated feature implies that BSP application can execute with other user applications in the same resource concurrently. The dynamic behavior deals with environment changes occurred at runtime (such as network congestion and alterations on processors load) and changes in processes, once some BSP processes can need more processing power or increase its network interaction with other ones during application execution.

The model of machine can include multiprocessors machines, local networks as well as clusters. The model requires that the involved nodes must have connections to allow all-to-all message passing. Each BSP process is mapped to real processor that is enclosed inside a node. In order to turn the scheduling more flexible and efficient, the load-rebalancing model used hierarchical scheduling. This model is based on a notion of hierarchy in two levels (local and global) that is present in the Integrade scheduler[2]. The vision of hierarchy is important to optimize the passing of monitoring information, such as processing and communication times as well as the processors load information. The nodes are aggregated to create an abstraction of a set. For example, a set should be a LAN network or a cluster. Each set has a manager that exchanges data with managers of other sets and it can be composed by one or more nodes.

## 2.2. Load Rebalancing Activation

The load-rebalancing through BSP processes migration is launched at the end of a superstep, after the barrier phase and before the next super-step. Thus, this mechanism answers the "When" question. This migration point was chosen because in this moment it is possible to analyze data from all BSP processes at their processing and communication phases. In this point, we have information about the slowest process, the amount of instructions performed by each one and the communication scheme among the processes. Aiming to generate as less intrusion in application as possible, it was used a technique that allows to adapt the interval of supersteps to call the load-rebalancing. The idea is to delay this call if the processes are balanced, *i. e.*, if they have the conclusion times of each super-step closely.

To turn viable the adaptivity on load-rebalancing calling, it was used an index $\alpha$ ($\alpha \in \mathbb{N} | \alpha \geq 1$) that informs the interval of supersteps adopted to apply the processes migration. This index increases if the system tends to the stability in conclusion time of each superstep and decreases case opposite. The last case means that the frequency of calls increases in order to turn the system more stable quickly. To permit a sliding $\alpha$, it is necessary to verify if the distributed system is balanced or not. To treat this issue, it is collected the time (processing added to communication) of each BSP process at the end of each superstep up to the $\alpha$ value to be achieved. After that, arithmetic average is computed according to these time values and the times of the slowest and fastest processes are captured. Using these values, it is possible to measure the distributed system balancing.

$$time\ of\ fastest\ process > average\ time\ .\ (1-D) \quad (1)$$

$$time\ of\ slowest\ process < average\ time\ .\ (1+D) \quad (2)$$

The distributed system is considered stable if both Inequalities 2 and 1 are true. In both inequalities, $D$ value informs the distance in percentage that the time of the slowest and the fastest process can be moved away from the average. The $D$ value is passed in load-rebalancing model initialization. Concerning this, Algorithm 1 shows how the $\alpha$ value is computed. Another variable called $\alpha$' was employed to save the temporary value of $\alpha$. Thus, $\alpha$' will indicate the next superstep interval to active the load-rebalancing. The $\alpha$' value suffers a variation of one unity depending on the state of the system: stable (balanced) or not.

**Algorithm 1** Interval of supersteps $\alpha$ for the next call to load-rebalancing of BSP application

```
1:  α' = α
2:  for From superstep k up to superstep k + α − 1 do
3:      if Inequalities 1 and 2 are true then
4:          Increase the α' value by 1
5:      else
6:          Decrease the α' value by 1 up arrive to 1
7:      end if
8:  end for
9:  Call for load-rebalancing of the BSP application
10: α = α'
```

In Algorithm 1, $k$ is the index of superstep that comes after the last call for load-rebalancing ($k$ is 1 if the model is beginning). The $\alpha$' does not have upper bound, but its lesser value is 1. In the best case, the system is always in equilibrium and $\alpha$' always increases. In architectural view, each set manager collects the execution times of its BSP processes at the end of each superstep. After that, each manager spreads its informations to the other managers using diffusion messages. When the interval of supersteps $\alpha$ is reached, each manager informs its processes to launch

the load-rebalancing. Thus, the load-rebalancing model is called by each BSP process that cooperate to decide which of them will be migrated.

### 2.3. Choosing Candidate Processes for Migration

We have used three metrics to choose those processes that will be migrated to new resources. The adopted metrics are Processing, Communication and Memory. They are employed to answer the "Which" question, that considers the BSP elected processes for migration. Processing metric obtains informations about the BSP local processing phase, while the Communication metric works with data collected in global communication phase. Memory metric enters in the model as an idea of cost and is used to measure the BSP process migration viability. These three metrics are combined to compute the Potential of Migration (PM) of each BSP process. Finally, this PM is used to select the candidate processes for migration.

**2.3.1. Processing Metric** Each BSP process $i$ computes the $P(i)$ function in Processing metric. To compute this function it is used data collected in a superstep $k$ up to $k+\alpha-1$ (where $k$ is the index of the first superstep after the calling for load-rebalancing). For each superstep $k$ in this interval it is stored the number of processor instructions ($I_k$) and the time for completion the processing phase ($PT_k$). The value of $I_k$ is used to evaluate the process stability (regularity) regarding the amount of instructions in each superstep. This stability is represented by the processing pattern called $P_{proc}$. This pattern is a real number enclosed in [0,1] interval. A $P_{proc}(i)$ closed to 1 means that the process $i$ is regular in the number of instructions that executes at each superstep. Its initial value is 1 for all processes, because it is made a bet that all process are stable.

**Algorithm 2** Processing Pattern $P_{proc}(i)$ of the process $i$

1: **for** From superstep $k$ up to superstep $k+\alpha-1$ **do**
2: **if** $P_k(i) \geq I_k(i).(1-\delta)$ and $P_k(i) \leq I_k(i).(1+\delta)$ **then**
3: Increase the $P_{proc}(i)$ by $\frac{1}{\alpha}$ up to 1
4: **else**
5: Decreases the $P_{proc}(i)$ by $\frac{1}{\alpha}$ down to 0
6: **end if**
7: **end for**

The $P_{proc}(i)$ of process $i$ increases or decreases depending on the prediction of the amount of performed instructions in each superstep. This prediction for the superstep $k$ and process $i$ is represented by $PI_k(i)$ and it is based on the aging concept. Following this scheme, a superstep depends on the data regarding itself and all previous supersteps until the first one after the load-rebalancing calling. The aging concept uses the idea that the prediction value is more strongly influenced by recent supersteps. The generic recurrence formula to compute the prediction $PI_k(i)$ is shown below. The value $i$ denotes a BSP process, $k$ a superstep index and $\alpha$ means the interval do activate the load-rebalancing.

- $PI_k(i) = I_k(i)$
- $PI_{k+1}(i) = \frac{1}{2}PI_k(i) + \frac{1}{2}I_{k+1}(i)$
- $PI_{k+\alpha-1}(i) = \frac{1}{2}PI_{k+\alpha-2}(i) + \frac{1}{2}I_{k+\alpha-1}(i)$

The advantage of this prediction scheme is that only data between two load-rebalancing activations (beetwen the supersteps $k$ and $k+\alpha-1$) is used. This scheme saves memory and contributes to decrease the prediction calculation time. On the other hand, the value of $P_{proc}(i)$ persists during the BSP application execution independently of the amount of calls for load-rebalancing. $P_{proc}(i)$ is updated following the Algorithm 2. We consider the system stable if the forecast is within a $\delta$ margin of fluctuation from the amount of instructions performed. For example, if $\delta$ is equal to 0.1 and the number of instructions is 50, the prediction must be between 45 and 55 to increase the $P_{proc}(i)$ value.

The processing pattern $P_{proc}(i)$ of process $i$ is an element in $P(i)$ function. The other element in $P(i)$ is a processing index called $IP(i)$. This index is derived from the processing time prediction $PTP_{k+\alpha-1}(i)$ of the process $i$ at the superstep $k+\alpha-1$ (the superstep where the load-rebalancing will be activated). Analogous to $PI$ prediction, $PTP$ also works with the aging concept. Supposing that $PT_k(i)$ is the processing time of the process $i$ during the superstep $k$, then the prediction $PTP_{k+\alpha-1}(i)$ is computed as follows.

- $PTP_k(i) = PT_k(i)$
- $PTP_{k+1}(i) = \frac{1}{2}PTP_k(i) + \frac{1}{2}PT_{k+1}(i)$
- $PTP_{k+\alpha-1}(i) = \frac{1}{2}PTP_{k+\alpha-2}(i) + \frac{1}{2}PT_{k+\alpha-1}(i)$

In order to compute the index $IP(i)$, each BSP process $i$ calculates its $PTP_{K+\alpha-1}$ prediction and passes it to its set manager. Each set manager transfers the prediction values to other managers. With data from all processes, each manager computes the values of $IP(i)$ of the processes under its responsibility through Equation 3. Firstly, the process that has the slowest prediction (the highest value) is captured. This process has $IP(i)$ equal to 1 and its $PTP_{k+\alpha-1}$ value enters as denominator in Equation 3. Therefore, the value of $IP(i)$ of the remaining processes are less than 1.

$$IP(i) = \frac{PTP_{k+\alpha-1}(i)}{\textit{the highest } PTP_{k+\alpha-1} \textit{ value}} \quad (3)$$

$$P(i) = P_{proc}(i) \; . \; IP(i) \quad (4)$$

Equation 4 shows the function to compute the Processing metric for process $i$. Its value is closed to 1 if the BSP process is stable in the number of instructions that it executes and if the considered process has a bigger processing time.

However, $P(i)$ is closed to 0 if the process is unstable (suffers large variations in the amount of instructions at each superstep) and/or it finishes its processing phase quickly.

**2.3.2. Communication Metric** The Communication metric is expressed through the function $C(i, j)$, where *i* denotes a BSP process and *j* the target set. This function treats the communication (sending and receiving) involving the process *i* and all processes that belong to the set *j*. To compute $C(i, j)$ it is used data collected in a superstep *k* up to $k + \alpha - 1$ (where *k* is the index of the first superstep after the calling for load-rebalancing). Besides this, each process maintain a table with *n* lines, where *n* is the amount of set in the distributed environment. Each table line has the following fields: (i) total time spent in communication with a specified set; (ii) communication pattern for this set, that is called $P_{com}(i, j)$. This pattern is a real number within the [0,1] interval and its alteration depending on the prediction $PC_k(i, j)$ that deals with the amount of bytes involved during communication between the process *i* and the set *j* at superstep *k*. Analogous to Processing metric, $PC_k(i, j)$ is based on aging concept and is organized as follows.

- $PC_k(i, j) = C_k(i, j)$
- $PC_{k+1}(i, j) = \frac{1}{2}PC_k(i, j) + \frac{1}{2}C_{k+1}(i, j)$
- $PC_{k+\alpha-1}(i, j) = \frac{1}{2}PC_{k+\alpha-2}(i, j) + \frac{1}{2}C_{K+\alpha-1}(i, j)$

In communication prediction context, $C_k(i, j)$ is a notation used to assign the number of communicated bytes between the process *i* and the set *j* at superstep *k*. The value of $C_k(i, j)$ is the amount of sent bytes if this value is higher than the received one and equal to the received bytes otherwise. The idea in this issue is to use the worst case in process and set communication. Using communication prediction, $Pcom(i, j)$ is computed according to Algorithm 3. This algorithm uses a variable $\beta$ that informs the acceptable variation in communication prediction.

**Algorithm 3** Communication Pattern $P_{com}(i, j)$

1: **for** From superstep *k* up to superstep $k + \alpha - 1$ **do**
2: **if** $(1 - \beta).C_k(i, j) \leq PC_k(i, j)$ e $(1 + \beta).C_k(i, j) \geq PC_k(i, j)$ **then**
3: Increases the value of $P_{com}(i, j)$ by $\frac{1}{\alpha}$ up to 1
4: **else**
5: Decreases the value of $P_{com}(i, j)$ by $\frac{1}{\alpha}$ down to 0
6: **end if**
7: **end for**

$P_{com}(i, j)$ is the first element in function $C(i, j)$. The second one is a communication index $IC(i, j)$ that is reached through the value of the prediction $PTC_{k+\alpha-1}(i, j)$. $PTC_{k+\alpha-1}(i, j)$ is the communication time prediction involving the process *i* and the set *j* at the superstep $k + \alpha - 1$. Adopting *k* as an index of the first superstep after the load-rebalancing calling, $k + \alpha - 1$ is the superstep index where it will be called again. In order to compute this prediction it is used the communication time $CT_k(i, j)$ between the process *i* and the set *j* at superstep *k*. It is transferring time if it is higher than the receiving one or equal to the receiving time otherwise. Concerning this, $PTC_{k+\alpha-1}(i, j)$ is achieved as follows.

- $PTC_k(i, j) = CT_k(i, j)$
- $PTC_{k+1}(i, j) = \frac{1}{2}PTC_k(i, j) + \frac{1}{2}CT_{k+1}(i, j)$
- $PTC_{k+\alpha-1}(i, j) = \frac{1}{2}PTC_{k+\alpha-2}(i, j) + \frac{1}{2}CT_{k+\alpha-1}(i, j)$

Each BSP process computes *n* functions $IC(i, j)$ using Equation 5, where *n* is the number of environment sets. Each BSP process *i* verifies the prediction $PTC_{k+\alpha-1}(i, j)$ which has the highest value (the slowest one) and this value enters as a denominator in Equation 5. The index $IC(i, j)$ that involves the process *i* and the set *j* of the highest prediction is set to 1. Therefore, the remaining values of $IC(i, j)$ for the same process *i* are smaller than 1. In this way, the function that represents the Communication metric between the process *i* and the set *j* is presented in Equation 6. Remembering that each process *i* computes *n* (number of sets) Equations 6 locally.

$$IP(i, j) = \frac{PTC_{k+\alpha-1}(i, j)}{\textit{the highest } PTC_{k+\alpha-1} \textit{ value}} \quad (5)$$

$$C(i, j) = P_{com}(i, j) \; . \; IC(i, j) \quad (6)$$

Equation 6 is to stay closed to 1 if the process *i* has a regularity considering the communicated bytes to processes of set *j* and performs slower communication actions to this set. The idea of communication is to use the highest value between sendings and receptions involving process *i* and set *j*. For example, a BSP process *i* can have a receptor characteristic of messages deriving from the processes that belong to set *j*. Thus, it is easier to migrate process *i* to other resource in set *j* than to migrate the involved processes in set *j* to the set where process *i* executes currently.

**2.3.3. Memory Metric** The function M(i,j) represents the Memory metric and evaluates the migration cost of the image of process *i* to another resource in set *j*. This metric just uses data collected at the superstep in which the load-rebalancing will be activated (where $\alpha$ is achieved). The first element of function $M(i, j)$ is a memory index called $IM(i)$. The BSP process with the biggest image memory in bytes has $IM(i)$ equal to 1. The indexes of the remaining processes are ordered based on this value and are lower than 1. Therefore, each $IM(i)$ index informs a comparative weight regarding the space in memory of each process.

The function $IM(i)$ is the first element used to compute the Memory metric. The other element is an index that treats the transferring cost of the process *i* to the set *j* and is symbolized by $IT(i, j)$. Firstly, each BSP process computes the time to transfer its memory image to the specified set using

a simple pingpong test. This time involves the network interaction between process *i* and manager of the set *j*. Case the destination set is the same one that the BSP process is located, the transferring time occurs between the target process and other resource of the same set randomly selected. The process *i* verifies the longest transfer time that computed and the element $IT(i,j)$ for this set *j* receives the value equal to 1. The indexes $IT(i,j)$ of other processes are organized taking as upper bound the previous computation and are lower than 1. For example, if the environment is composed by three sets and the transferring time of process *i* to them are 10, 20 and 15 seconds, respectively. Applying the algorithm, $IT(i,1)$ is equal to 0.5, $IT(i,2)$ is equal to 1 and $IT(i,3)$ is 0.75. Combining both elements, we compute Memory metric using Equation 7.

$$M(i,j) = IM(i) \; . \; IT(i,j) \tag{7}$$

Analyzing Memory metric, each BSP process will compute *n* $M(i,j)$, where *n* is the number of sets in the environment. A $IM(i,j)$ value closed to 1 means a higher migration cost and informs a BSP process *i* with large memory (if compared with the remaining ones) and a longer transferring time of its image memory to the target set. Nevertheless, $M(i,j)$ near to 0 specifies a process with small memory image and/or shorter transferring time to the target set.

**2.3.4. Potential of Migration Analysis** Aiming to generate the Potential of Migration (PM) of each process, we are using the notion of force from the physics area. In physics, force is an influence that may make an object to accelerate and can be represented by a vector. A vector has a size (magnitude), a direction and an angle of actuation. Analyzing force idea, each studied metric can be seen as a vector that acts over an object. In our case, this object is the migration of a BSP process. Vectors $\vec{P}$ and $\vec{C}$ represent the Processing and Communication metrics, respectively. Both vectors have the same angle and direction and stimulate the process migration. On the other side, the Memory metric means the migration cost and is symbolized by vector $\vec{M}$. This vector works against the migration, once it has the same angle but a different direction if compared with $\vec{P}$ and $\vec{C}$ vectors. Observing the forces that act over the process migration, the resultant informs the Potential of Migration and is reached using Equation 8.

$$P\vec{M} = \vec{P} + \vec{C} - \vec{M} \tag{8}$$

$$PM(i,j) = P(i) + C(i,j) - M(i,j) \tag{9}$$

In the load-rebalancing model context, the Potential of Migration of a process *i* to the set *j* is denoted by $PM(i,j)$ and is computed by the Equation 9. P, C and M represent the Processing, Communication and Memory metrics, respectively. Considering that each metric can vary between 0 and 1, the $PM(i,j)$ value is within [-1,2] interval. The maximum value is achieved when P and C metrics have values equal to 1 and the M metric is null. The lower bound of $PM(i,j)$ occurs when there is a high migration cost and a null force to stimulate the migration. Each process *i* will compute *n* equations 9, where *n* is the amount of sets. After that, process *i* sends its highest Potential of Migration to its set manager. The load-rebalancing model uses a migration threshold that selects which processes will be migrated. For instance, a threshold equals to 1.5 implies that only processes with PM above this value will migrate.

### 2.4. Analyzing Destination of Elected Processes

The BSP process migration happens after the barrier synchronization of superstep which $\alpha$ is reached (see subsection 2.3). A elected process *i* has a targeted set *j* informed in its Potential of Migration $PM(i,j)$. Thus, the pertinent question is to choose which node of this set will be the destination of the BSP process and, therefore, to answer the question "Where" of the load-rebalancing model. Firstly, the manager set of process *i* contacts the manager of the set *j* asking it for a physical processor to receive a process. This last manager verifies the resources under its responsibility and elects the destination processor (and the node).

$$ICPU(p) = peak(p) \; . \; (1 - load(p)) \tag{10}$$

The manager of destination set selects a processor based on the Equation 10 that evaluates the CPU index (ICPU) of each processor. In this equation, *p* means the physical processor and the function $peak(p)$ informs the performance peak achieved by processor *p*. The function $load(p)$ in the same equation represents the CPU *p* average load in the last 15 minutes. This time interval was adopted based on studies from Vozmediano e Conde [8]. The manager sorts the indexes $ICPU(p)$ decreasingly in a list. After that, it captures the first element of the list and verifies if the node that belongs the observed processor has available memory to store the new BSP process. Case this node does not have memory, the manager analyzes the next element until this memory issue is solved. If this issue was solved, the processor *p* of the current $ICPU(p)$ is chosen as a destination. On the other hand, if the sorted list was fully observed and the memory issue was not solved, the manager set returns an error code that means the migration is failed. In this scheme, the communication issue was not used. The communication cost among nodes inside the same set is considered uniform.

## 3. Implementation Issues

The model implementation is an on going work. We are working to implement a prototype of a system for load-rebalancing of BSP processes that can be integrated to message passing library. In this context, we are studying libraries that implement MPI interface and offer processes migration. There is a initiative in GPPD group that aims to offer processes migration in MPI applications that will be

considered in load-rebalancing system. Other possibility is to use AMPI (Adaptive MPI)[5] library that employs the Charm++ tool to turn viable the migration. There are some technical issues that must be offered in order to implement the prototype. In processing part, the prototype needs data about the amount of performed instructions and the processing time at each superstep on each process. In communication part, the prototype needs to analyze the amount of sent or received bytes to/from each set, as well as the conclusion time of these communication actions.

## 4. Related Work

The transferring of autonomous objects in WAN environment is presented in [7]. This WAN environment is composed by a set of interconnected LAN networks and their load-balancing mechanism just considers informations from the communication actions and network equipment features. Other work is the migration performed by the GridWay resource broker, that treats with time and cost optimization scheduling and migration[8]. Both migration mechanisms consider data from CPU, like speed and load. Kondo et al. [6] described a client-server scheduling model for global computing applications. Their model uses the processor speed, the network bandwidth and disk space metrics to determine the number of work units that can be sent to a client. However, these values are not combined and the minimum of them gives the amount of unit works.

Bhandarkar, Brunner and Kale[1] presented a runtime support for adaptive load-rebalancing in MPI applications using process migration. Periodically, the MPI application transfers control to the load balancer using a special call MPI_Migrate(), which allows the framework to invoke a strategy to remap processes to new processors. These authors present a Metis-based strategy that uses the communication graph to remap the processes. As well as Vozmediano e Conde work[8], this work uses a fixed threshold to define those process that will be migrated. Vadhiyar e Dongarra[9] presented a migration framework and self adaptivity feature in GrADS system. They computed the cost of each migration (15 minutes) and the rescheduling gain based on the remaining execution time prediction over a new specified resource. In order to turn possible this computation, this framework must work with well know applications, where its parts and duration are known. In order to measure the migration cost at application runtime, Du, Sun and Wu[4] described a migration model that considers the process, the memory, the I/O and the communication states. However, these authors did not specify neither when to launch the process migration, nor which processes will be migrated.

The PUBWCL[3] is a library that offers parallel algorithms according to BSP model. This library aims to use the idle processing power of Internet distributed computers. In order to offer load-rebalancing, PUBWCL can migrate BSP process during barrier synchronization phase. This mechanism considers the processing phase and does not take into account the communication among the processes. Besides processing and communication features of a superstep, our load-rebalancing model also includes the the migration cost idea (represented by Memory metric).

## 5. Conclusion

The load-rebalancing is activated at the end of a superstep and occurs according to system stability. The processes remapping is delayed if the BSP processes are balanced. In order to decide which processes will migrate, we developed an idea of Potential of Migration (PM). PM equation considers a BSP process and a destination set and is found using the Processing, Communication and Memory metrics. These three metrics are combined using an analogy of force from physics area. Processing and communication metrics act as forces that stimulate the process migration, while the Memory one (migration cost) works in contrary way. Only those processes that have PM higher than a specified threshold will be migrated. Finally, the destination is selected based on elected PM that informs the targeted set. The manager of this set chooses a better resource following a evaluation function.

## References


[1] M. A. Bhandarkar and L. V. Kal. Run-time support for adaptive load balancing. In *IPDPS '00*, pages 1152–1159, 2000.

[2] C. Boeres, A. P. Nascimento, V. E. F. Rebello, and A. C. Sena. Efficient hierarchical self-scheduling for mpi applications executing in computational grids. In *MGC '05: Workshop on Middleware for grid computing*, pages 1–6, New York, 2005.

[3] O. Bonorden, J. Gehweiler, and F. Meyer auf der Heide. Load balancing strategies in a web computing environment. In *International Conference on Parallel Processing and Applied Mathematics* , pages 839–846, 2005.

[4] C. Du, X.-H. Sun, and M. Wu. Dynamic scheduling with process migration. In *CCGRID '07:* pages 92–99, 2007.

[5] C. Huang, G. Zheng and S. Kumar. Performance evaluation of adaptive mpi. In *PPoPP '06: Symposium on Principles and practice of parallel programming*, pages 12–21, 2006.

[6] D. Kondo, H. Casanova, E. Wing, and F. Berman. Models and scheduling mechanisms for global computing applications. In *IPDPS '02: International Symposium on Parallel and Distributed Processing*, page 79-87, 2002.

[7] N. Krivokapic, M. Islinger, and A. Kemper. Migrating autonomous objects in a wan environment. *Jorunal of Intelligent Information Systems*, 15(3):221–251, 2000.

[8] R. Moreno-Vozmediano and A. B. Alonso-Conde. Influence of grid economic factors on scheduling and migration. In *High Performance Computing for Computational Science - VECPAR*, pages 274–287, 2005.

[9] S. S. Vadhiyar and J. J. Dongarra. Self adaptivity in grid computing: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(2-4):235–257, 2005.