Self-Organizing the Selection of Migratable Processes on Cluster-of-Clusters Environments

Rodrigo da Rosa Righi, Lucas Graebin, Rafael Bohrer Ávila
Programa Interdisciplinar de Pós-Graduação em Computação Aplicada – UNISINOS – São Leopoldo - Brazil
Email: rrrighi@unisinos.br, lgraebin@acm.com, rbavila@unisinos.br
Philippe Olivier Alexandre Navaux, Laércio Lima Pilla
Programa De Pós-Graduação em Computação – UFRGS – Porto Alegre - Brazil
Email: {navaux, llpilla}@inf.ufrgs.br

***Abstract*—The decision to move processes to new resources is NP-Hard, and heuristics take place in order to reach good results inside an acceptable time interval. In this way, this paper presents AutoMig — a novel heuristic for BSP applications that self-organizes the selection of candidates for migration on different clusters. Its differential approach consists in a prediction function (*pf*) that considers both processes' computation and communication data as well as their migration costs. *pf* is applied over a list of schedules and AutoMig's final step decides whether one of them outperforms the time of the current mapping. The results emphasize gains up to 32% when testing a CPU-bound application in a simulated cluster-of-clusters environment. Besides AutoMig, this paper also describes the rescheduling model associated with it.**

***Keywords*-BSP, rescheduling, heuristic, self-organizing.**

I. INTRODUCTION

Generally, process migration is implemented within the application with explicit calls [11]. A different migration approach happens at middleware level, where changes in the application code and previous knowledge about the system are usually not required. Considering this, we have developed a process rescheduling model called MigBSP [4]. It was designed to work with round-based applications with a BSP behavior (Bulk Synchronous Parallel). Concerning the choosing of the processes, MigBSP creates a priority list based on the highest Potential of Migration (*PM*) of each process [4]. *PM* combines the migration costs with data from computation and communication phases in order to create an unified scheduling metric. The process denoted on the top of the list is selected to be inspected for migration. Although we achieved good results with this approach, we agree that a selection of a percentage of processes could determine better results. However, a question arises: How can one reach an optimized value for dynamic environments? A solution involves the testing of several hand-tuned parameters and a comparison among the results.

After developing the first version of MigBSP, we have observed the promotion of intelligent scheduling systems which adjust their parameters on the fly and hide intrinsic optimization decisions from users [11]. In this context, we developed a new heuristic named **AutoMig** that selects one or more candidates for migration automatically. We took advantage of both List Scheduling and Backtracking concepts to evaluate the migration impact on each element of the *PM* list in an autonomous fashion. In addition, another AutoMig's strength comprises the needlessness to provide an additional parameter in MigBSP for getting more than one migratable process on rescheduling activation. The scheduling evaluation uses a prediction function (*pf*) that considers the migration costs and works following the concept of a BSP superstep [1]. The lowest pf indicates the most suitable rescheduling plan.

This paper aims to describe AutoMig in details. We evaluated it by using an BSP application for image compression [7]. Considering that the programmer does not need to change his/her application nor add a parameter in MigBSP, the results with migration were satisfactory and totaled a mean gain of 7.9%. This index was observed when comparing migrations with the application execution solely. The results also showed a serie of situations where AutoMig outperforms the heuristic that elects only one process.

We organized the paper in eight sections. Section 2 presents MigBSP. The main part of the paper belongs to Section 3, where Automig is described in details. Sections 4 and 5 show the employed methodology and the results, respectively. Related work is discussed in Section 6, while Section 7 presents the conclusion and future work. Finally, Section 8 shows our acknowledgments to Brazilian agencies.

II. MIGBSP: RESCHEDULING MODEL

MigBSP is a rescheduling model that works over heterogeneous resources, joining the power of clusters, supercomputers and local networks. The heterogeneity issue considers the processors' clock (all processors have the same set of instructions), as well as network bandwidth. Such an architecture is assembled with Sets (sites) and Set Managers. Set Managers are responsible for scheduling, capturing data from a Set and exchanging it among other managers.

The decision for process remapping is taken at the end of a superstep. Aiming to generate the least intrusiveness in application as possible, we applied two adaptations that control the value of α ($\alpha \in \mathbb{N}^*$). α is updated at each rescheduling call and will indicate the interval for the next

 ISBN: 978-1-61208-161-8

one. The adaptations' objectives are: (i) to postpone the rescheduling call if the processes are balanced or to turn it more frequent, otherwise; (ii) to delay this call if a pattern without migrations on ω past calls is observed. A variable named D is used to indicate a percentage of how far the slowest and the fastest processes may be from the average to consider the processes as balanced.

The answer for "Which" is solved through our decision function called Potential of Migration (*PM*). Each process i computes n functions $PM(i,j)$, where n is the number of Sets and j means a Set. The key rationale consists in performing only a subset of the processes-resources tests at the rescheduling moment. $PM(i,j)$ is found using Computation, Communication and Memory metrics as we can see in Equations 1, 2, 3 and 4. A previous paper describes each equation in details [4]. The greater the value of $PM(i,j)$, the more prone the processes will be to migrate.

$$Comp(i,j) = P_{comp}(i)\ .\ CTP(i)\ .\ ISet(j) \quad (1)$$

$$Comm(i,j) = P_{comm}(i,j)\ .\ BTP(i,j) \quad (2)$$

$$Mem(i,j) = M(i)\ .\ T(i,j)\ +\ Mig(i,j) \quad (3)$$

$$PM(i,j) = Comp(i,j) + Comm(i,j) - Mem(i,j) \quad (4)$$

Computation metric - $Comp(i,j)$ - considers a Computation Pattern $P_{comp}(i)$ that measures the stability of a process i regarding the amount of instructions at each superstep. This value is close to 1 if the process is regular and close to 0 otherwise. This metric also performs a computation time prediction $CTP(i)$ for process i based on all computation phases between two rescheduling activations. $Comp(i,j)$ also presents an index $ISet(j)$ which informs the average capacity of Set j. In the same way, Communication metric – $Comm(i,j)$ – computes the Communication Pattern $P_{comm}(i,j)$ between processes and Sets. Furthermore, this metric uses communication time prediction $BTP(i,j)$ considering data between two rebalancing activations. $Comm(i,j)$ increases if process i has a regular communication with processes from Set j and performs slower communication actions to this Set. Memory metric – $Mem(i,j)$ – considers process memory, transferring rate between considered process and the manager of target Set, as well as migration costs. These costs are dependent of the operating system, as well as the migration tool.

At each rescheduling call, each process passes its highest $PM(i,j)$ to its Set Manager. This last entity exchanges the *PM* of the processes with other managers. Each manager creates a decreasing-sorted list and selects the process on the top for testing the migration viability. This test considers the following data: (i) the external load on source and destination processors; (ii) the processes that both processors are executing; (iii) the simulation of considered process running on a destination processor; (iv) the time of communication actions considering local and destination processors; (v) migration costs. We computed two times: t_1 and t_2. t_1 means the local execution of process i, while t_2 encompasses its execution on the other processor and includes the migration costs. A new resource is chosen if $t_1 > t_2$.

III. AutoMig: A Novel Heuristic to Select the Suitable Processes for Migration

AutoMig's self-organizes the migratable processes without programmer intervention. It can elect not only one but a collection of processes at the migration moment. Especially, AutoMig's proposal solves the problem described below.

- **Problem Statement** - Given n BSP processes and a list of the highest *PM* of each one at the migration moment, the challenge consists in creating and evaluating at maximum n new scheduling plans and to choose the most profitable one among those that outperform the current processes-resources mapping.

AutoMig solves this question by using the concepts from List Scheduling and Backtracking. Firstly, we sort the *PM* list in a decreasing-ordered manner. Thus, the tests begin by the process on the head since its rescheduling represents better chances of migration gains. Secondly, AutoMig proposes n scheduling attempts (where n is the number of processes) by incrementing the movement of only one process at each new plan. This idea is based on the Backtracking functioning, where each partial candidate is the parent of candidates that differ from it by a single extension step. Figure 1 depicts an example of this approach, where a single migration on level l causes an impact on $l+1$. For instance, the performance forecast for process "A" considers its own migration and the fact that "E" and "B" were migrated too. Algorithm 1 presents AutoMig's approach in details.

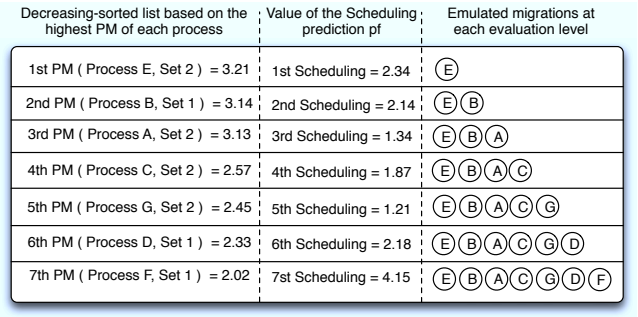

Figure 1. Example of the AutoMig's approach

The main part of AutoMig concerns its prediction function *pf*. *pf* emulates the time of a superstep by analyzing the computation and communication parts of the processes. Both parts are computed through Equations 5 and 6, respectively. They work with data collected at the superstep before calling the rescheduling facility. In addition, *pf* considers information about the migration costs of the processes to the

 ISBN: 978-1-61208-161-8

Sets. The final selection of migratable processes is obtained through verifying the lowest *pf*. The processes in the level belonging to this prediction are elected for migration if their rescheduling outperforms the *pf* for the current mapping.

At the rescheduling call, each process passes the following data to its manager: (i) its highest *PM*; (ii) a vector with its migration costs (Mem metric) for each Set; (iii) the number of instructions; (iv) a vector which contains the number of bytes involved on communication actions to each Set. Each manager exchanges *PM* values and uses them to create a decreasing-sorted list. Task 5 of Algorithm 1 is responsible for getting data to evaluate the current scheduling.

At each level of the *PM* list, the data of the target process is transferred to the destination Set. For instance, data from process 'E' is transferred to Set 2 according to the example illustrated in Figure 1. Thus, the manager on the destination Set will choose a suitable processor for the process and will calculate Equations 5 and 6 for it. Aiming to minimize multicast communication among the managers at each *pf* computation, each Set Manager computes $Time_p$ and $Comm_p$ for the processes under its jurisdiction and save the results together with the specific level of the list. After performing the tasks for each element on *PM* list, the managers exchange vectors and compute *pf* for each level as well as for the present scheduling (task 12 in Algorithm 1).

Equation 5 computes $Time_p(i)$, where i means a specific process. $Time_p(i)$ uses data related to the computing power and the load of the processor in which process i executes currently or is being tested for rescheduling. $cpu\_load(i)$ represents the CPU load average on the last 15 minutes. This time interval was adopted based on work of Vozmediano and Conde [9]. Equation 6 presents how we get the maximum communication time when considering process i and Set j. In this context, Set j may be the current Set of process i or a Set in which this process is being evaluated for migration. $T(k,j)$ refers to the transferring rate of 1 byte from the Set Manager of Set j to other Set Manager. $Bytes(i,k)$ works with the number of bytes transferred through the network among process i and all process belonging to Set k. Lastly, $Mig\_Costs(i,j)$ denotes the migration costs related to the sending of process i to Set j. It receives the value of the Mem metric, which also considers a process i and a Set j.

$$Time_p(i) = \frac{Instruction(i)}{(1 - cpu\_load(i)).cpu(i)} \quad (5)$$

$$\begin{aligned} Comm_p(i,j) &= Max_k\ (\ \forall\ k \in Sets \\ &\quad (Bytes(i,k)\ .\ T(k,j))\ ) \end{aligned} \quad (6)$$

$$\begin{aligned} pf &= Max_i\ (Time_p(i)) + Max_{i,j}\ (Comm_p(i,j)) \\ &+ Max_{i,j}\ (Mig\_Costs(i,j)) \end{aligned} \quad (7)$$

Considering Equation 7, we can emphasize that each part may consider a different process i and Set j. For instance, a specific process may obtain the largest computation time, while other one expends more time in communication actions. AutoMig uses a global strategy, where data from all processes are considered in the calculus. We take profit from the barriers of the BSP model for exchanging scheduling data, not paying additional costs for that.

Algorithm 1 AutoMig's approach for selecting the processes

1: Each process computes *PM* locally (see Equation 4).
2: Each process passes its highest *PM*, together with the number of instructions and a vector that describes its communication actions, to the Set Manager.
3: Set Managers exchange *PM* data of their processes.
4: Set Managers create a sorted list based on the *PM* values with n elements (n is the number of processes).
5: Set Managers compute Eq. 5 and 6 for their processes. The results will be used later for measuring the current mapping. Migrations costs are not considered.
6: **for** each element from 0 up to $n-1$ in the *PM* list **do**
7: Considered element is analyzed. Set Manager of process i sends data about it to the Set Manager of Set j. The algorithm proceeds its calculus by considering that process i is passed to Set j.
8: The manager on the destination Set chooses a suitable processor to receive the candidate process i.
9: Set Managers compute Eq. 5 and 6 for their processes.
10: Set Managers save the results in a vector with the specific level of the *PM* list.
11: **end for**
12: Set Managers exchange data and compute *pf* for the current scheduling as well as for each level on *PM* list.
13: **if** $Min(pf)$ in the *PM* list $<$ current *pf* **then**
14: Considering the *PM* list, the processes in the level where *pf* was reached are selected for migration.
15: Managers notify their elected processes to migrate.
16: **else**
17: Migrations do not take place.
18: **end if**

IV. Evaluation Methodology

We are simulating the functioning of a BSP-based Fractal Image Compression (FIC) application [7]. FIC applications apply transformations which approximate smaller parts of the image by larger ones. The smaller parts are called ranges and the larger ones domains. All ranges together form the image. The domains can be selected freely within the image. A complete domain-poll of an image of size $t \times t$ with square domains of size $d \times d$ consists of $(t-d+1)^2$ domains. Furthermore, each domain has 8 isometries. So each range is compared with $8(t-d+1)^2$ domains. The application time increases as the number of domains increases as well. Our BSP modeling considers the variation of both the range and domain sizes as well as the number of processes. Algorithm

2 presents the organization of a single superstep. Firstly, we are computing $\frac{t}{r}$ supersteps, where $t \times t$ is the image size and r is the size of square ranges. The goal is to compute a set of ranges at each superstep. For that, each superstep works over $\frac{t}{r}$ ranges since the image comprises a square. At each superstep, a range is computed against $8((\frac{t}{d})^2.\frac{1}{n})$ domains, where d represents the size of a domain and n the number of processes. Thus, each process sends $\frac{t}{r}$ ranges before calling the barrier, which must be multiplied by 8 to find the number of bytes (each range occupies 8 bytes).

Algorithm 2 Single superstep for FIC problem

```
1: Taking a range-pool rp (0 ≤ rp ≤ t/r − 1): t and r mean
   the sides of the t × t image and r × r range, respectively
2: for each range in rp do
3:    for each domain belonging to specific process do
4:       for each isometry of a domain do
5:          calculate-rms(range, domain)
6:       end for
7:    end for
8: end for
9: Each process i (0 ≤ i ≤ n − 1) sends data to its right-
   neighbor i + 1. Process n − 1 sends data to process 0
10: Call for synchronization barrier
```

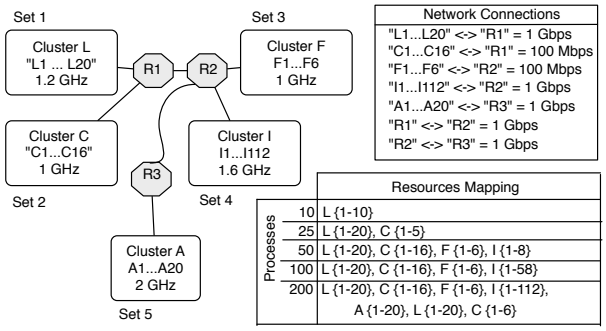

Figure 2. Multiple Clusters-based topology, processing and network resources description and the initial processes-resources scheduling

The BSP application was evaluated with simulation in three scenarios: (i) Application execution simply; (ii) Application execution with MigBSP scheduler without applying migrations; (iii) Application execution with MigBSP scheduler allowing migrations. Both the application and AutoMig were developed using the SimGrid Simulator (MSG Module) [3]. It is deterministic, where a specific input always results in the same output. The scenarios were evaluated in an infrastructure with five Sets (see Figure 2). A Set represents a cluster where each node has a single processor.

The infrastructure permits us to analyze the impact of the heterogeneity issue on AutoMig's algorithms.

Initial tests were executed using α equal to 4 and D equal to 0.5. We observed the behavior of 10, 25, 50, 100 and 200 BSP processes. Their initial mapping to the resources may be viewed in Figure 2. Since the application proceeds in communications from process i to $i+1$, we are using the contiguous approach in which a cluster is filled before passing to another one [10]. The values of 40, 20 and 10 were used for the side (d) of a square domain and the figure is a square 1000x1000. The lower the d value, the greater the number of domains for computation. Finally, the migration costs are based on tests with AMPI in our clusters.

V. Analyzing AutoMig's Overhead and Decisions

Table I presents the tests with 40 and 20 for both domain and range sizes, respectively. This setup enables a small computation grain and processes migrations are not viable. *PM* values in all situations are negative, owing to the lower weight of the computation and communication actions if compared to the migration costs. AutoMig figures out the lowest *pf* for the current scheduling. Thus, both times for scenario ii and iii are higher than scenario i. In this context, a large overhead is imposed by MigBSP since the normal application execution is close to 1 second in average.

Table I
Results with 40 for domain (time in seconds)

Processes	Scenario i	Scenario ii	Scenario iii
10	1.20	2.17	2.17
25	0.66	1.96	1.96
50	0.57	2.06	2.06
100	0.93	2.44	2.44
200	1.74	3.41	3.41

We increase the number of domains when dealing with 20 for the domain's side. The execution with 20 for domain is depicted in Figure 3. The execution with 10 processes did not present replacement because they are balanced. *pf* of 0.21 was obtained for the current processes-resources mapping by using 20 for domain and 10 processes. All predictions in the *PM* list are higher than 0.21 and their average achieves 0.38. However, this configuration of domain triggers migration when using 25 and 50 processes. In the former case, 5 processes from cluster C are moved to the fastest cluster named A. AutoMig's decisions led a gain of 17.15% with process rescheduling in this context. The last mentioned cluster receives all processes from cluster F when dealing with 50 processes. This situation shows up gains of 12.05% with migrations. All processes from cluster C remain on their initial location because the computation grain decreases with 50 processes. Although 14 nodes in the fastest cluster A stay free, AutoMig does not select some processes for execution on them because the BSP model presents a barrier. Despite 14 migrations from cluster C to A occur, a group of process in the slower cluster will remain inside it and still limit the superstep's time. Lastly, since the work grain decreases when adding more processes, the executions with 100 and 200 did not present migrations.

 ISBN: 978-1-61208-161-8

We achieved better results when using 10 for domain (see Figure 4). The computation grain increases exponentially with this configuration. This sentence may be viewed through the execution of 10 processes, in which are all migrated to cluster A. Considering that $8((\frac{t}{d})^2 \cdot \frac{1}{10})$ express the number of domains assigned to each one of 10 processes, this expression is equal to 500, 2000 and 8000 when testing 40, 20 and 10 values for domain. Using 10 for both domain and the number of processes, the current scheduling produced a *pf* of 1.62. *pf* for the *PM* list is shown below:

- $pf[1..10] = \{1.79, 1.75, 1.78, 1.79, 1.81, 1.76, 1.74, 1.82, 1.78, 1.47\}$.

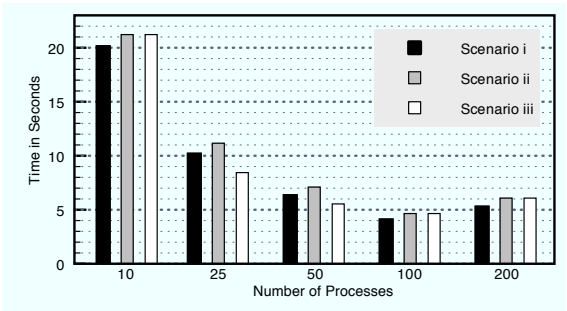

Figure 3. AutoMig's evaluation when using 20 for domain

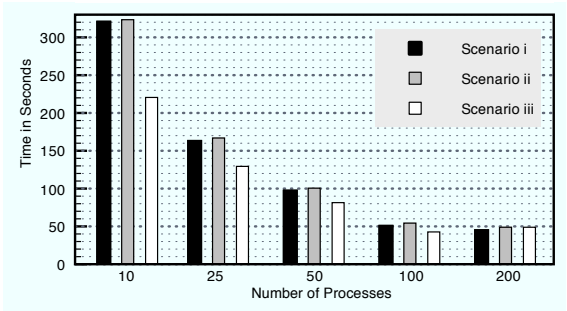

Figure 4. AutoMig's results when enlarging the work per process at each superstep. This graph illustrates experiments with domain 10 and range 5

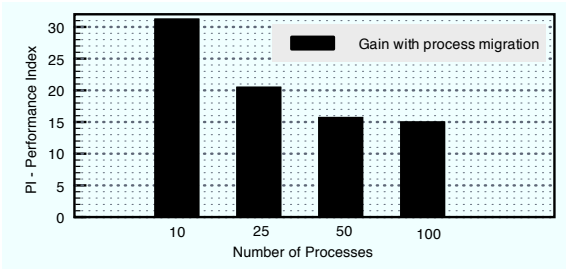

Figure 5. Migration gains with domain 10. PI =$(\frac{scen.\ i - scen.\ iii}{scen.\ i} * 100)$

Considering the first up to the ninth *pf* in the last itemization, we observed that although some processes can run faster in a more appropriate cluster, there are others that remain in a slower cluster. This last group does not allow performance gains due to the BSP modeling. However, the migration of 10 processes to the fastest cluster generates a *pf* of 1.47 and a gain around 31.13% when comparing scenarios iii and i. This analysis is illustrated in Figure 5. The processes from cluster C are moved to A with 25 processes and domain equal to 10. In this case, the 20 other processes stay on cluster L because there are not enough free nodes in the fastest cluster. A possibility is to explore two process in a node of cluster A (each node has 2 GHz) but AutoMig does not apply it because each node in Cluster L has 1.2 GHz. Considering the growth in the number of domains, the migrations with 100 processes becomes viable and get 14.95% of profit. Nevertheless, the initial mapping of 200 processes stands the same and an overhead of 7.64% was observed when comparing both scenarios i and ii.

We can conclude that the higher the computation weight per process, the better will be the gains with process rescheduling. In this way, we tested AutoMig with a shorter domain as expressed in Table II. This table shows the behavior for 10 and 25 processes. Gains about 31.62% and 19.81% were obtained when dealing with AutoMig. In addition, its overhead is shorter than 1%. We verified that the benefits with migrations remain practically constant if we compare the executions with 10 and 4 for the domain values. It is possible to observe that when doubling the number of processes, the application time is not halved as well.

Table II
EXECUTION TIME (IN SECONDS) WITH DOMAIN 4

Processes	Scen.i	Old Heuristic		AutoMig	
		Scen.ii	Scen.iii	Scen.ii	Scen.iii
10	12500.51	12511.87	9191.72	12523.22	8555.29
25	6250.49	6257.18	5311.54	6265.38	5011.77

Table II also shows a comparative analysis of the two selection heuristics implemented in MigBSP. We named the one that selects one process at each rescheduling call as Old Heuristic. Despite both obtained good levels of performance, AutoMig achieves better migration results than Old Heuristic (approximately 8%). For instance, 5 processes are migrated already in the first attempt for migration when testing 25 processes. In this case, all processes that were running on Cluster C are passed to Cluster A. This reorganization suggested by AutoMig at the beginning of the application provides a shorter time for application conclusion. In the other hand, 5 rescheduling calls are needed to reach the same configuration expressed previously with Old Heuristic. Lastly, AutoMig imposes larger overheads if compared to Old Heuristic (close to 1%). This situation was expected since two multicast communications among the Set Managers are performed by AutoMig in its algorithms.

VI. RELATED WORK

Vadhiyar and Dongarra presented a migration framework and self-adaptivity in GrADS system [12]. The gain with rescheduling is based on the remaining execution time prediction over a new specified resource. This framework must work with applications in which their parts and durations are known in advance. Sanjay and Vadhiyar [11] present

a scheduling algorithm called Box Elimination. It considers a 3-D box of CPU, bandwidth and processors tuples for selecting the resources with minimum available CPU and bandwidth. This work treats applications in which the problem size is known in advance. Liu et al. [8] introduced a novel algorithm for resource selection. The application reports the Execution Satisfaction Degree (ESD) to the scheduling middleware. The main weakness of this idea is the fact that users/developers need to define the ESD function by themselves for each new application.

Concerning the migration context, PUBWCL [2] and PUB [1] libraries enable this facility on BSP applications. PUBWCL aims to take profit of idle cycles from nodes around the Internet [2]. All proposed algorithms just use data about the computation times of each process as well as data regarding the nodes's load. The PUB's author proposed both centralized and distributed strategies for load balancing. Both strategies consider neither the communication among the processes, nor the migration costs.

VII. Conclusion and Future Work

Considering that the bulk synchronous style is a common organization for MPI programs [1], [6], AutoMig emerges as an alternative for selecting their processes for running on more suitable resources without interferences from the developers. AutoMig's main contribution appears on its prediction function *pf*. *pf* is applied for the current scheduling as well as for each level of a Potential of Migration-based list. Each element of this list informs a new scheduling through the increment of one process replacement. *pf* considers the load on both the Sets and the network, estimates the slowest processes regarding their computation and communication actions and adds the migration costs. The key problem to solve may be summarized in maintaining the current processes' location or to choose a level of the list.

AutoMig's load balancing scheme uses the global approach, where data from all processes are considered in the calculus [13]. Instead to pay a synchronization cost to get the scheduling information, AutoMig takes profit from the BSP superstep concept in which a barrier always occurs after communication actions. AutoMig and an application were developed using the SimGrid Simulator. Since the application is CPU-bound, the shorter the domain's size the higher the application's time and migration profitability. The results proved this, indicating gains up to 17.15% and 31.13% for domains equal 20 and 10. Particularly, the results revealed the main AutoMig's strength on selecting the migratable processes. It can elect the whole set of processes belonging to a slower cluster to run faster in a more appropriate one. But, sometimes a faster cluster has fewer free nodes than the number of candidates. Migrations do not take place in this situation owing to the execution rules of a BSP superstep.

Finally, future work comprises the use of AutoMig in a HPC service for Cloud computing. Concerning that each application specifies its own SLA previously, AutoMig appears as the first initiative to reorganize the processes-resources shaping when SLA fails. If the rescheduling does not solve the problem, more resources are allocated in a second instance.

VIII. Acknowledgments

This work is partially supported by CTIC RNP and CNPq.

References

[1] O. Bonorden. Load balancing in the bulk-synchronous-parallel setting using process migrations. In *21th International Parallel and Distributed Processing Symposium (IPDPS 2007)*, pages 1–9. IEEE, 2007.

[2] O. Bonorden, J. Gehweiler, and F. M. auf der Heide. Load balancing strategies in a web computing environment. In *Proceeedings of International Conference on Parallel Processing and Applied Mathematics (PPAM)*, pages 839–846, Poznan, Poland, September 2005.

[3] H. Casanova, A. Legrand, and M. Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Int. Conf. on Computer Modeling and Simulation (uksim)*, pages 126–131, 2008. IEEE.

[4] R. da Rosa Righi, L. L. Pilla, A. Carissimi, P. A. Navaux, and H.-U. Heiss. Observing the impact of multiple metrics and runtime adaptations on bsp process rescheduling. *Parallel Processing Letters*, 20(2):123–144, June 2010.

[5] F. J. da Silva, F. Kon, A. Goldman, M. Finger, R. Y. de Camargo, F. C. Filho, and F. M. Costa. Application execution management on the integrade opportunistic grid middleware. *J. Parallel Distrib. Comput.*, 70(5):573–583, 2010.

[6] R. E. De Grande and A. Boukerche. Dynamic balancing of communication and computation load for hla-based simulations on large-scale distributed systems. *J. Parallel Distrib. Comput.*, 71:40–52, 2011.

[7] Y. Guo, X. Chen, M. Deng, Z. Wang, W. Lv, C. Xu, and T. Wang. The fractal compression coding in mobile video monitoring system. In *CMC '09: Proceedings of the 2009 WRI International Conference on Communications and Mobile Computing*, pages 492–495, Washington, DC, USA, 2009. IEEE Computer Society.

[8] H. Liu, S.-A. Sørensen, and A. Nazir. On-line automatic resource selection in distributed computing. In *Int. Conf. on Cluster Computing*, pages 1–9. IEEE, 2009.

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

[10] J. A. Pascual, J. Navaridas, and J. Miguel-Alonso. Job scheduling strategies for parallel processing. chapter Effects of Topology-Aware Allocation Policies on Scheduling Performance, pages 138–156. Springer, 2009.

[11] H. A. Sanjay and S. S. Vadhiyar. A strategy for scheduling tightly coupled parallel applications on clusters. *Concurr. Comput. : Pract. Exper.*, 21(18):2491–2517, 2009.

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

[13] M. J. Zaki, W. Li, and S. Parthasarathy. Customized dynamic load balancing for a network of workstations. *J. Parallel Distrib. Comput.*, 43(2):156–162, 1997.

 ISBN: 978-1-61208-161-8